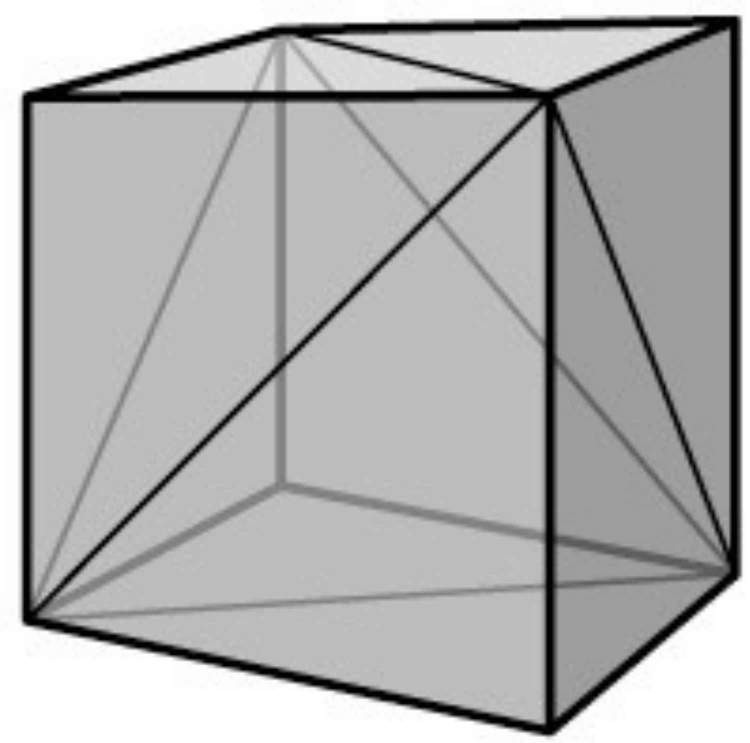
Lecture 7:

Digital Geometry Processing

Interactive Computer Graphics
Stanford CS248, Winter 2019

A small triangle mesh

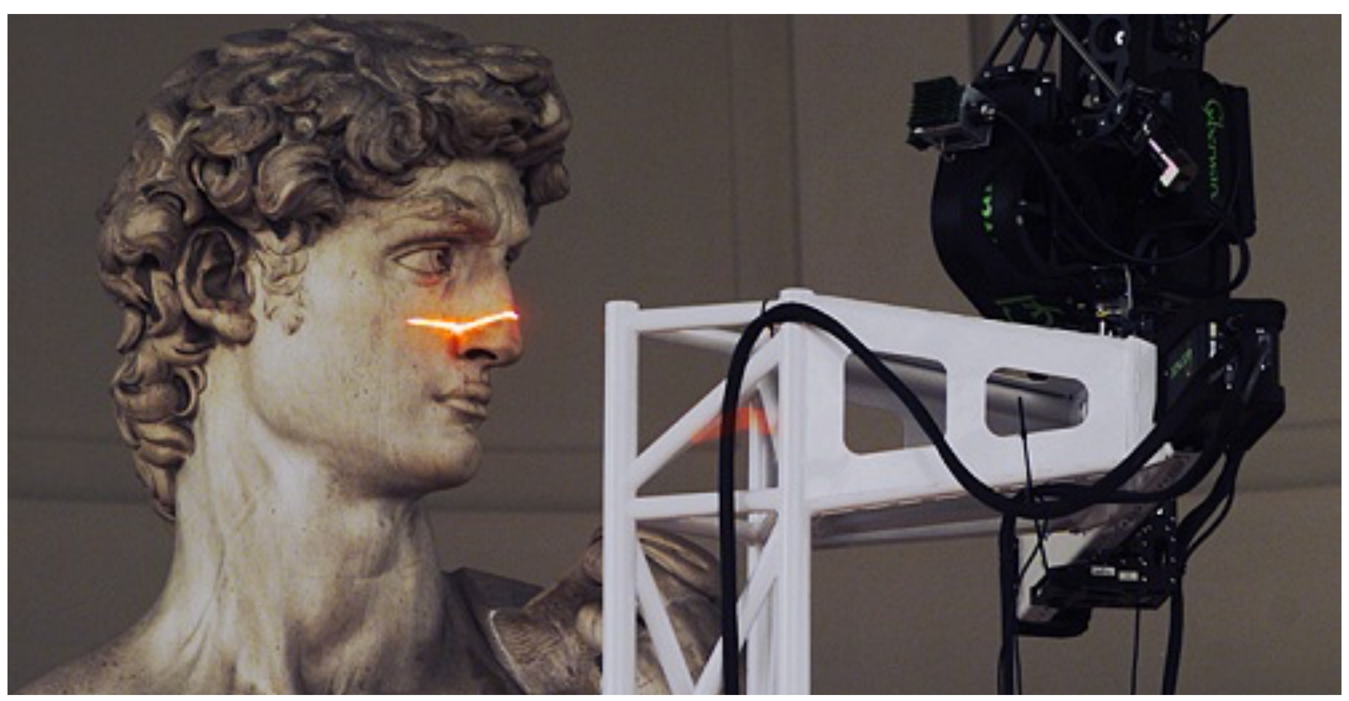


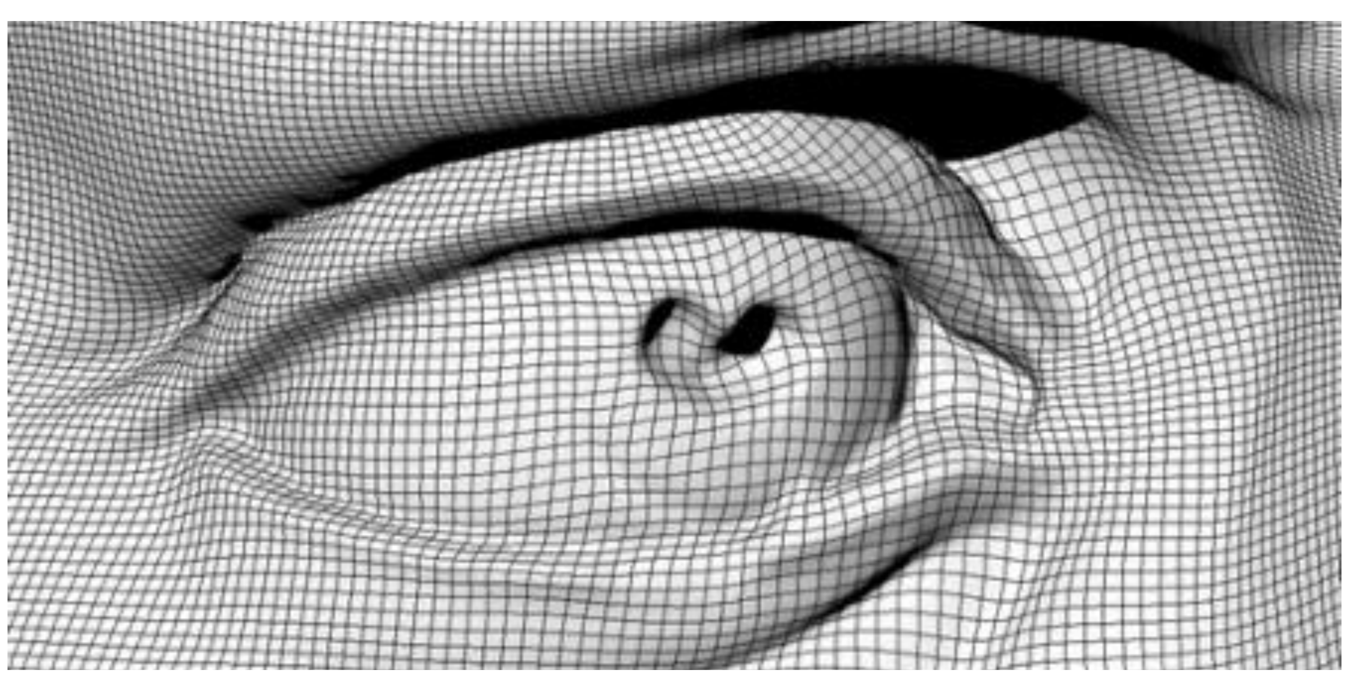
8 vertices, 12 triangles

A large triangle mesh

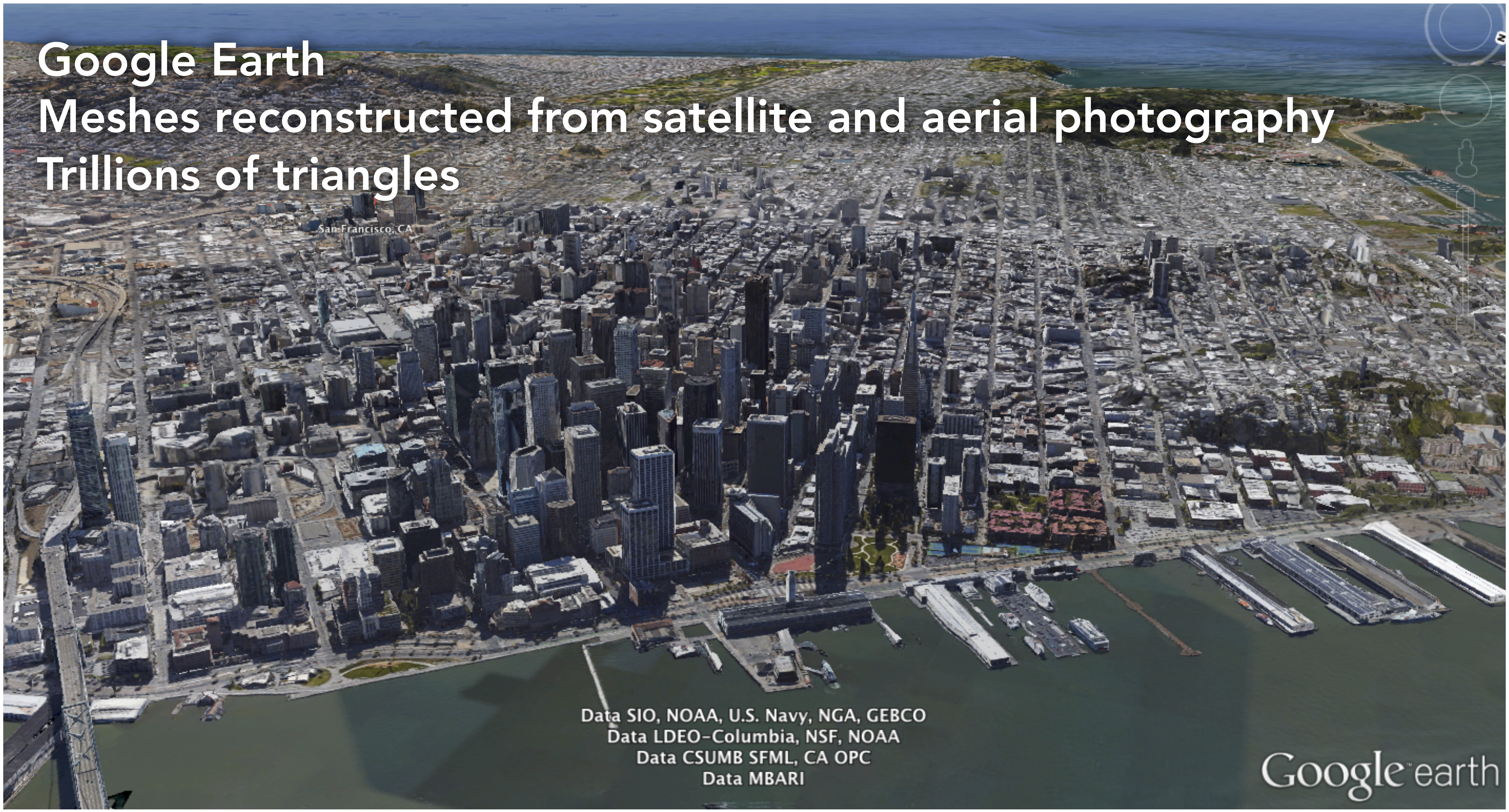
David

Digital Michelangelo Project
28,184,526 vertices
56,230,343 triangles



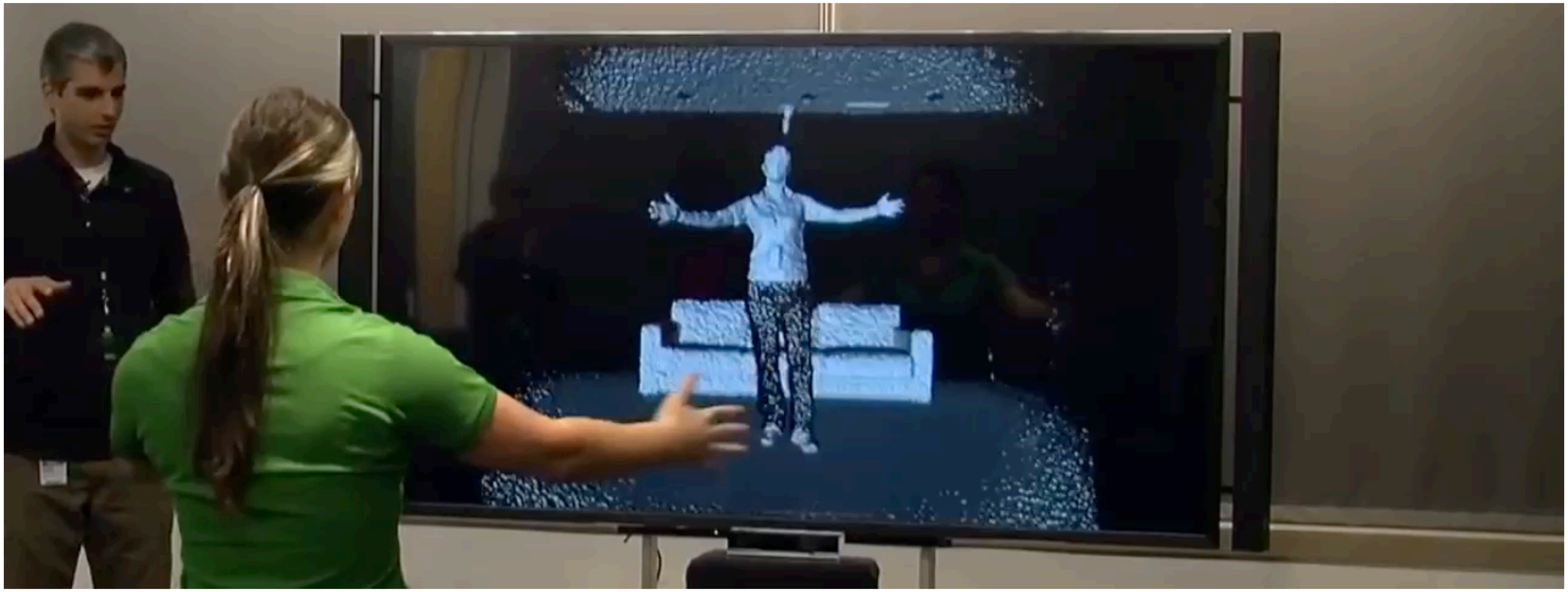


Even larger meshes

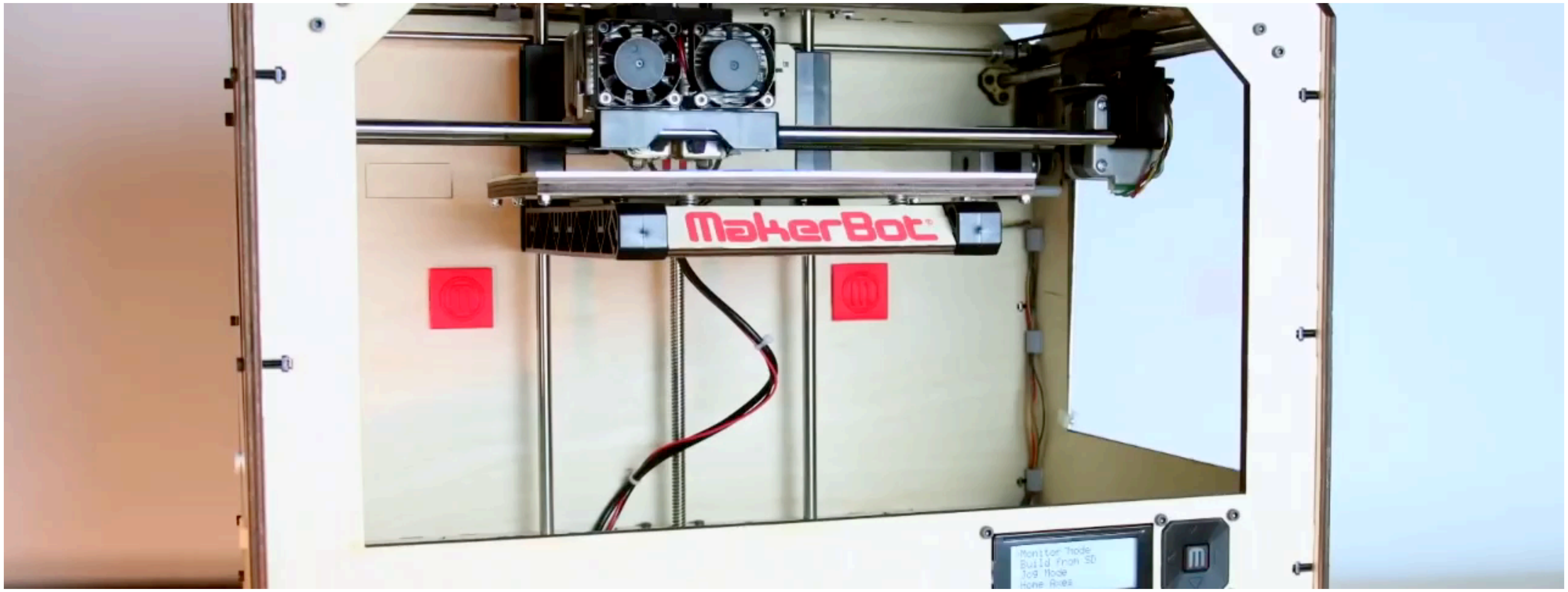


Digital geometry processing: motivations

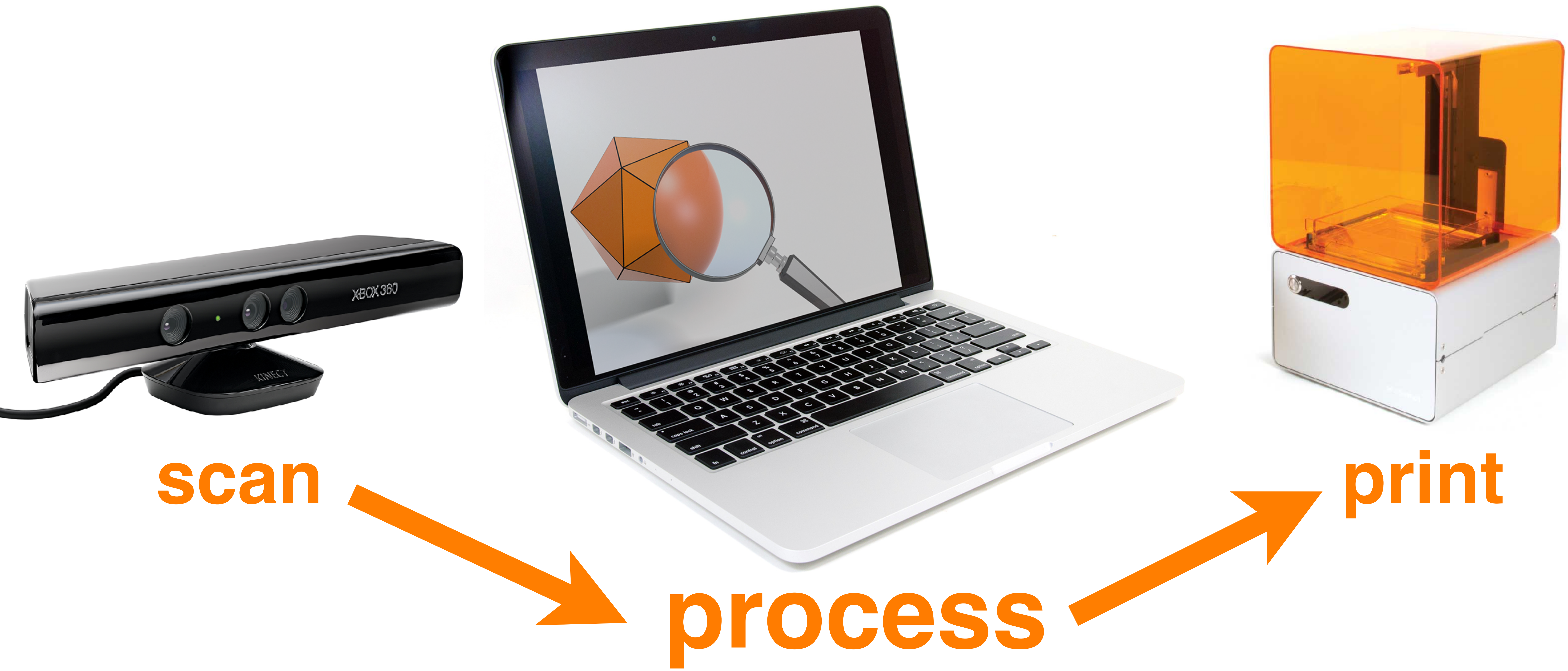
3D Scanning



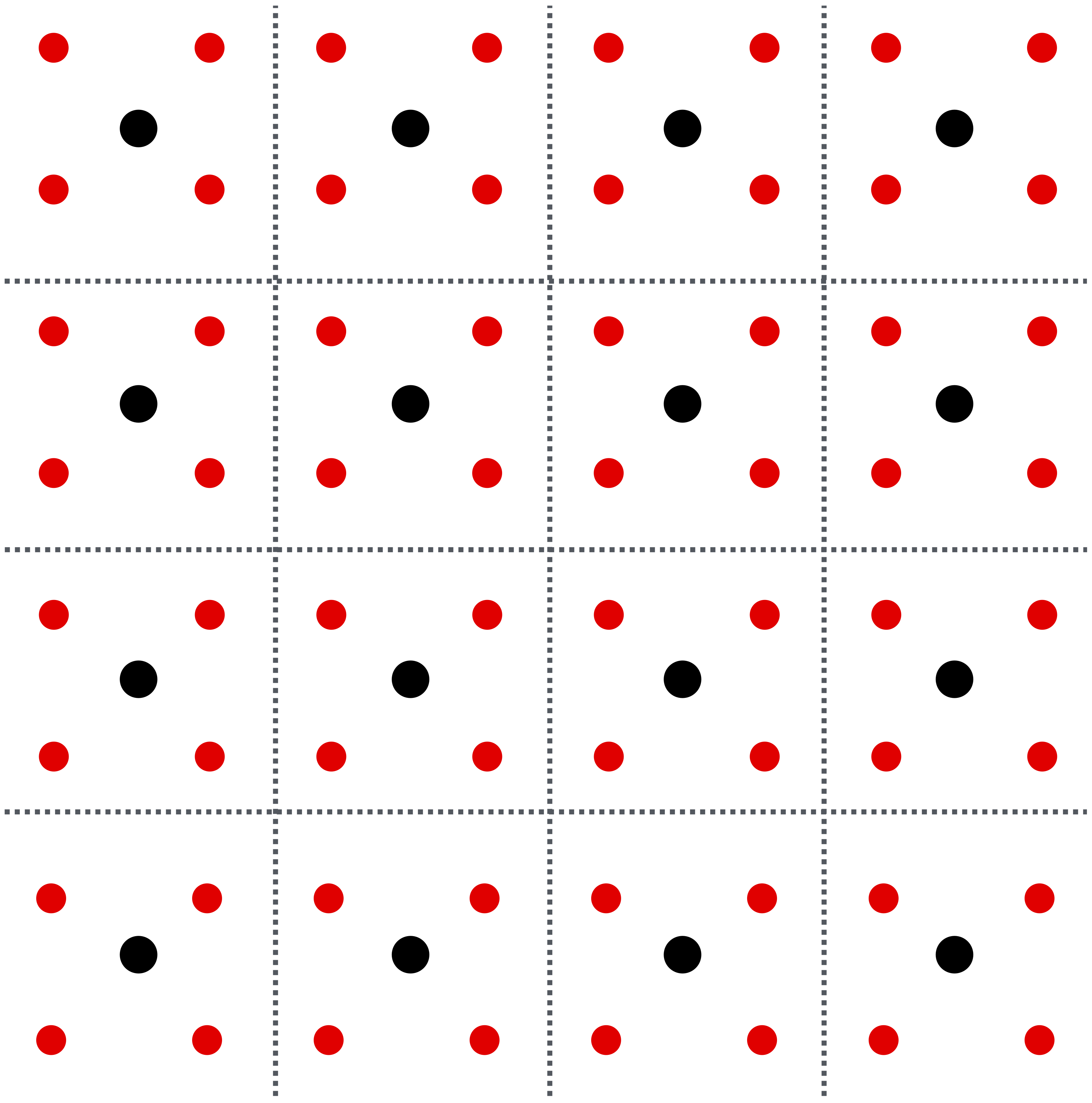
3D Printing



Geometry processing pipeline



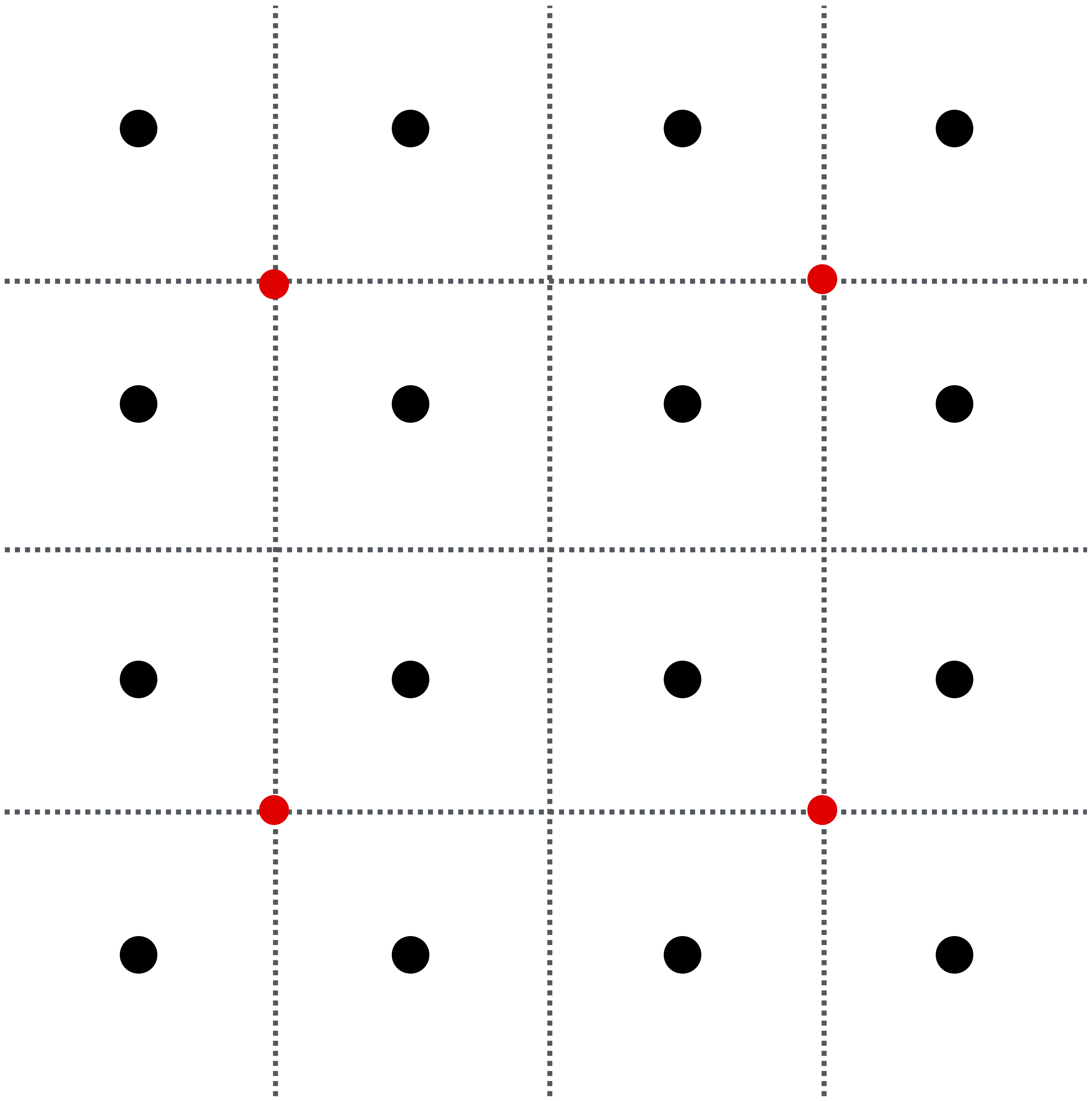
Recall: image upsampling



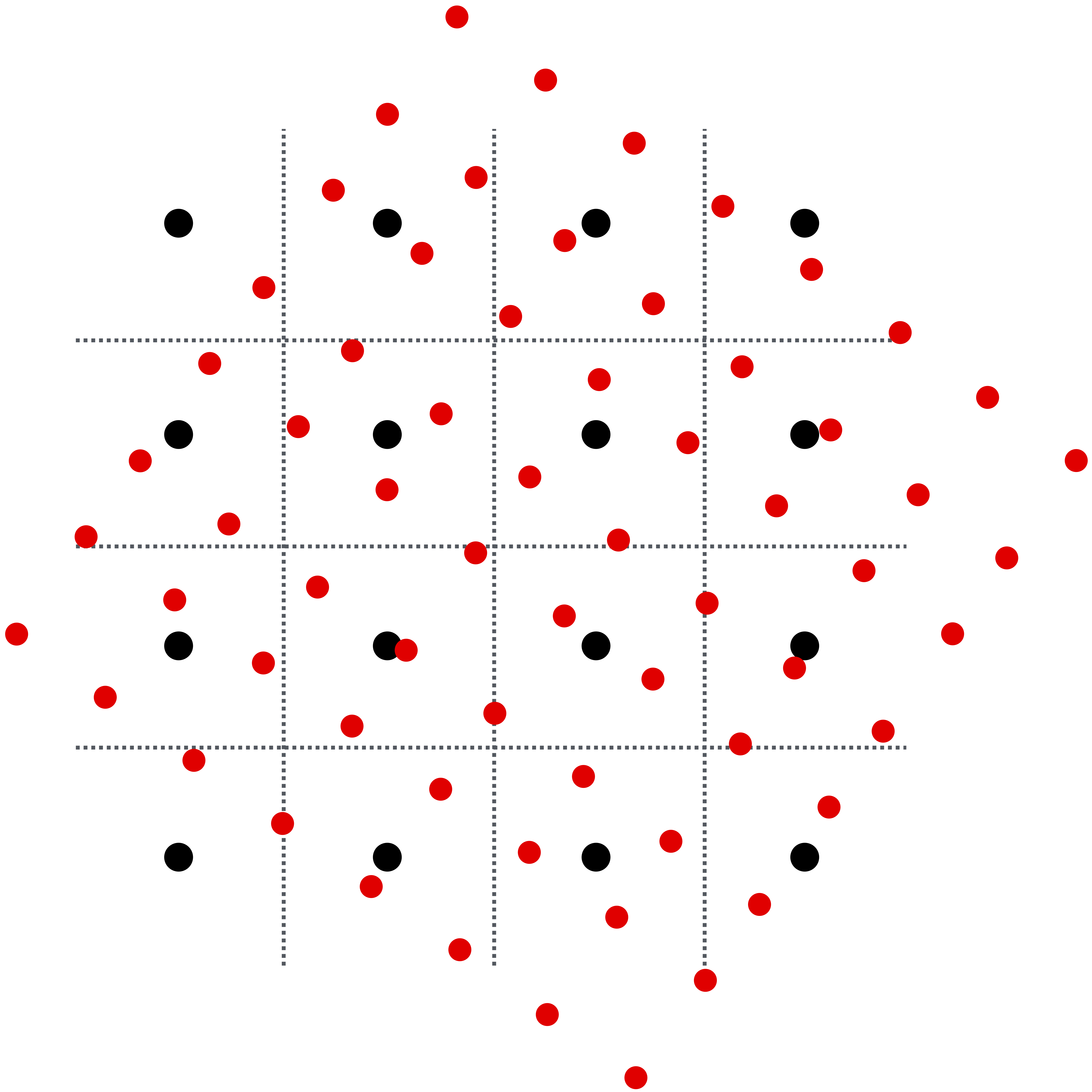
Recall: image upsampling

Upsampling via bilinear interpolation

Recall: image downsampling

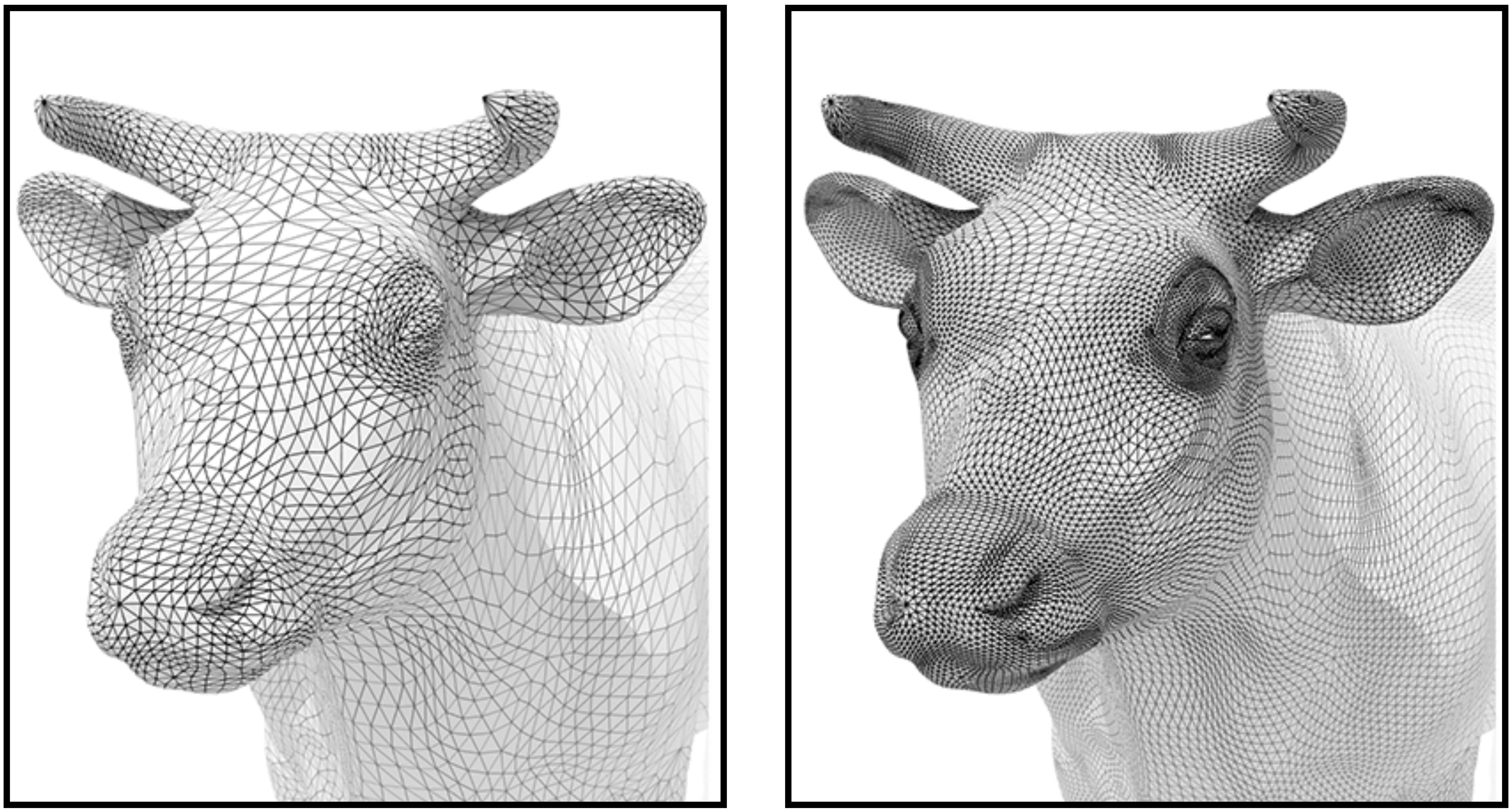


Recall: image resampling



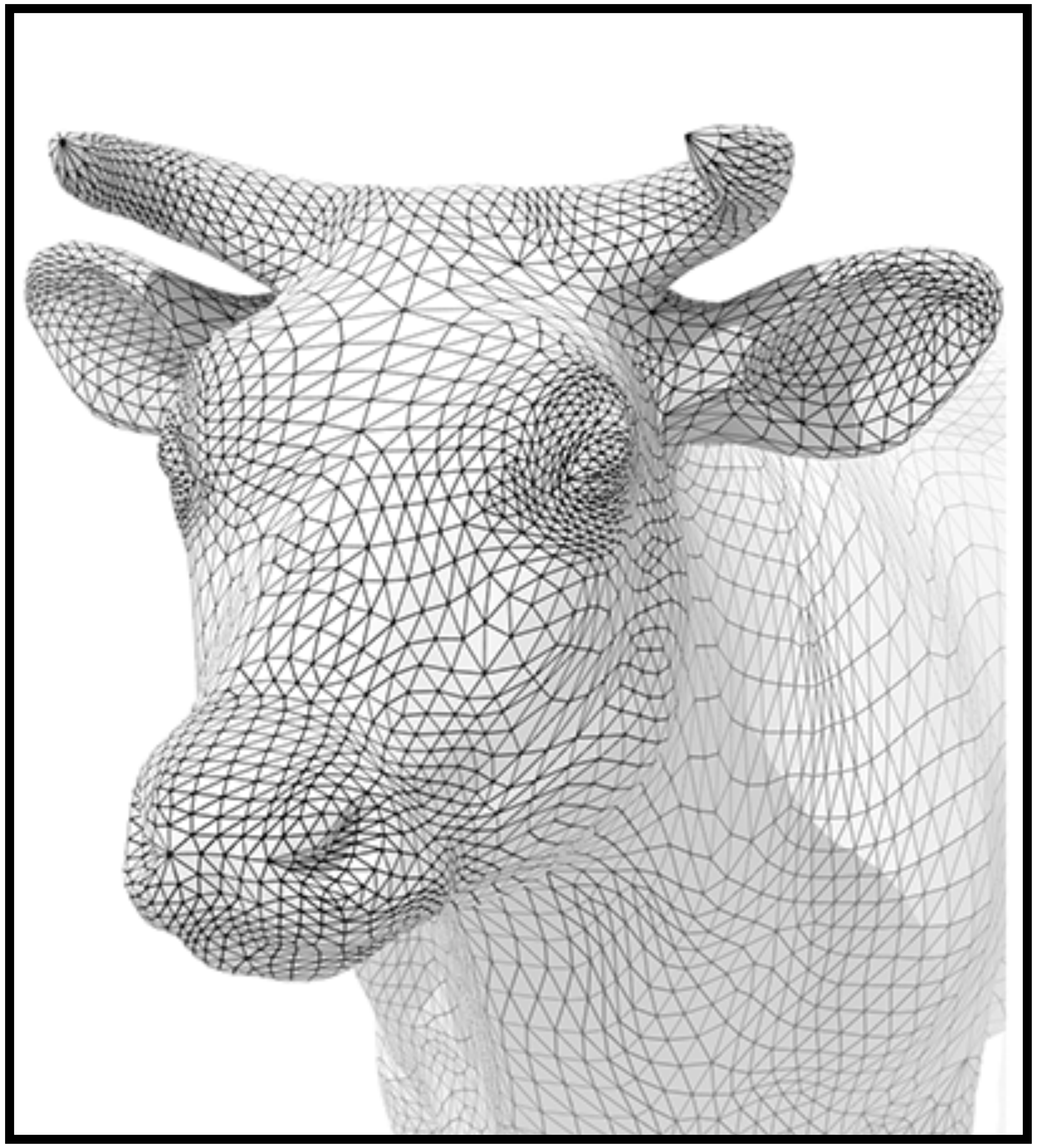
Examples of geometry processing

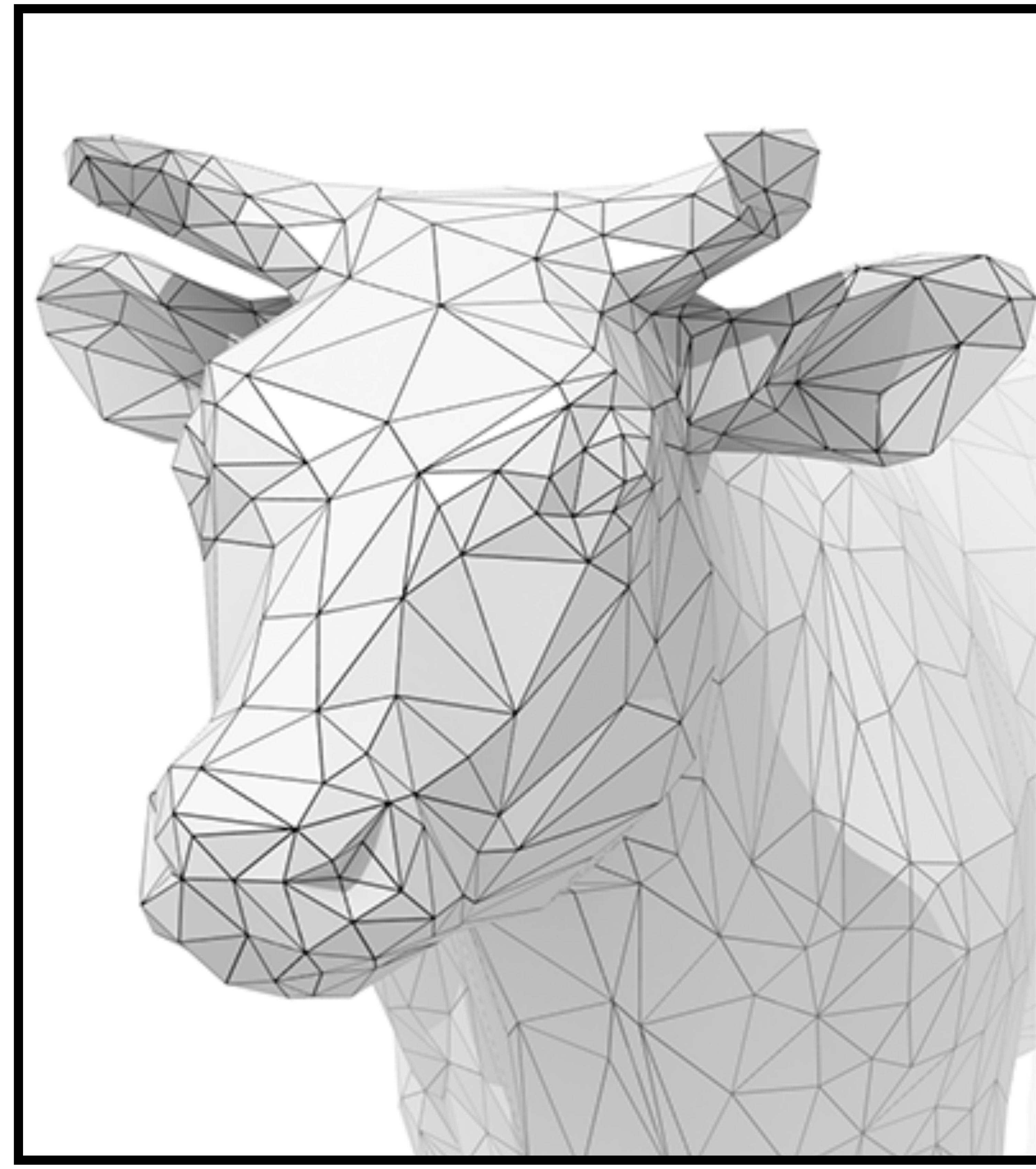
Mesh upsampling — subdivision



Increase resolution via interpolation

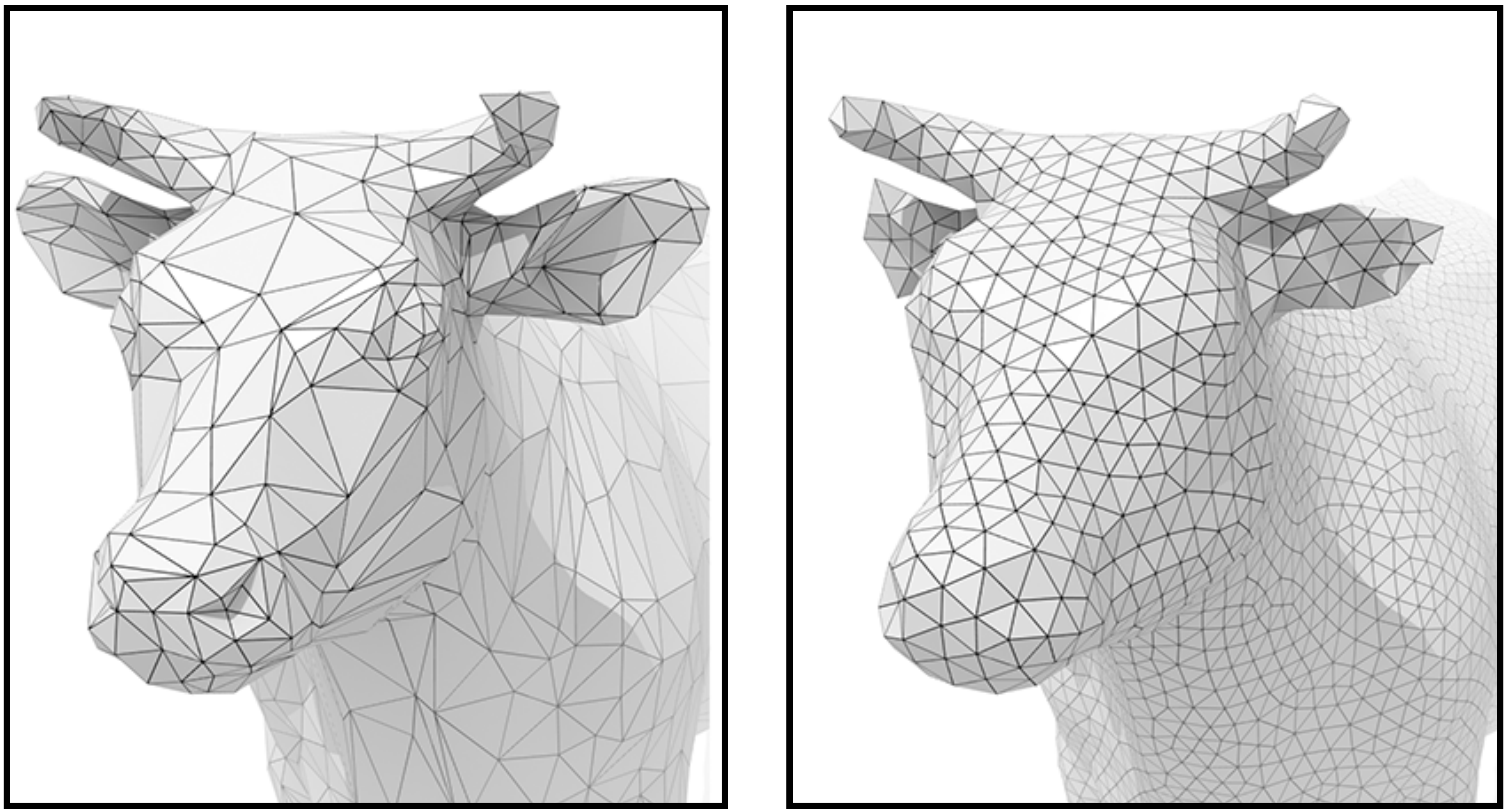
Mesh downsampling — simplification





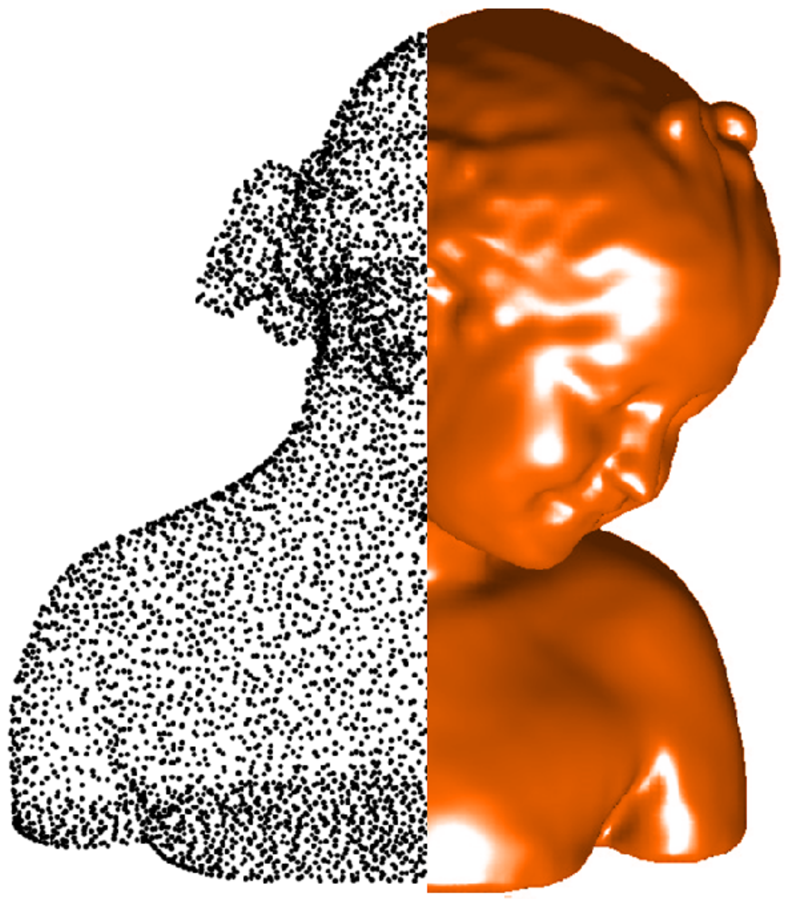
Decrease resolution; try to preserve shape/appearance

Mesh resampling — regularization



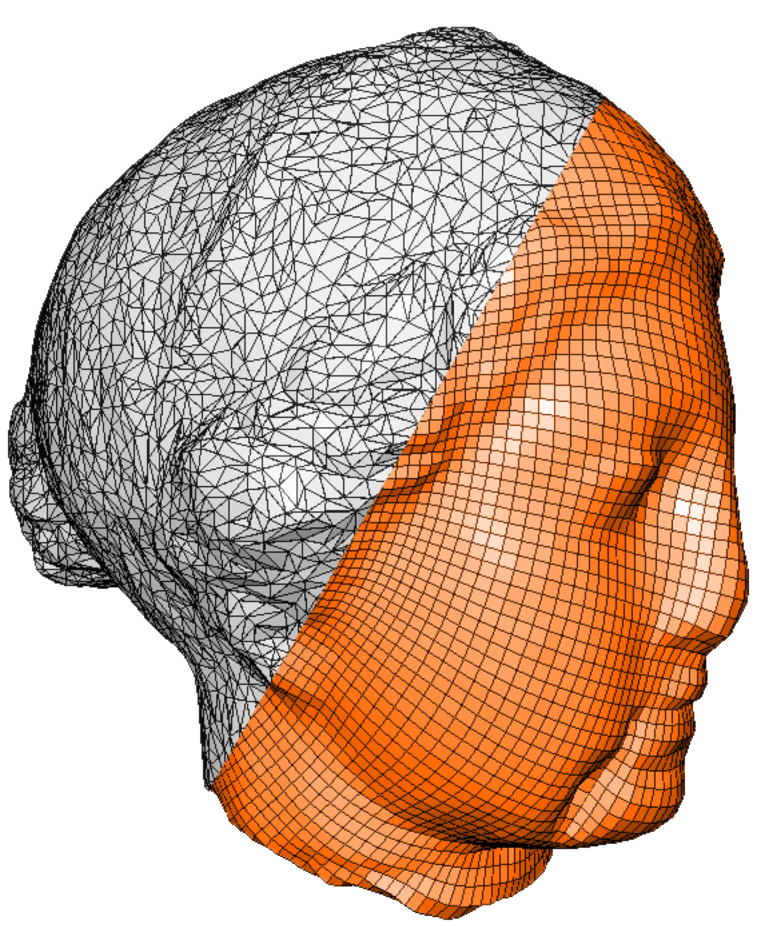
Modify sample distribution to improve quality

More geometry processing tasks

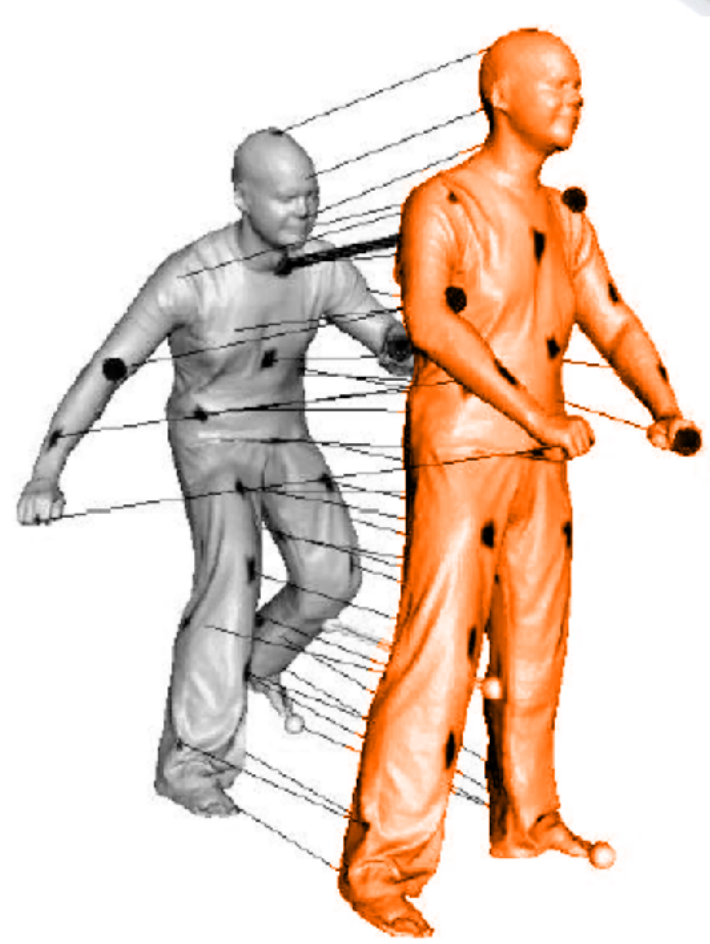


reconstruction

filtering

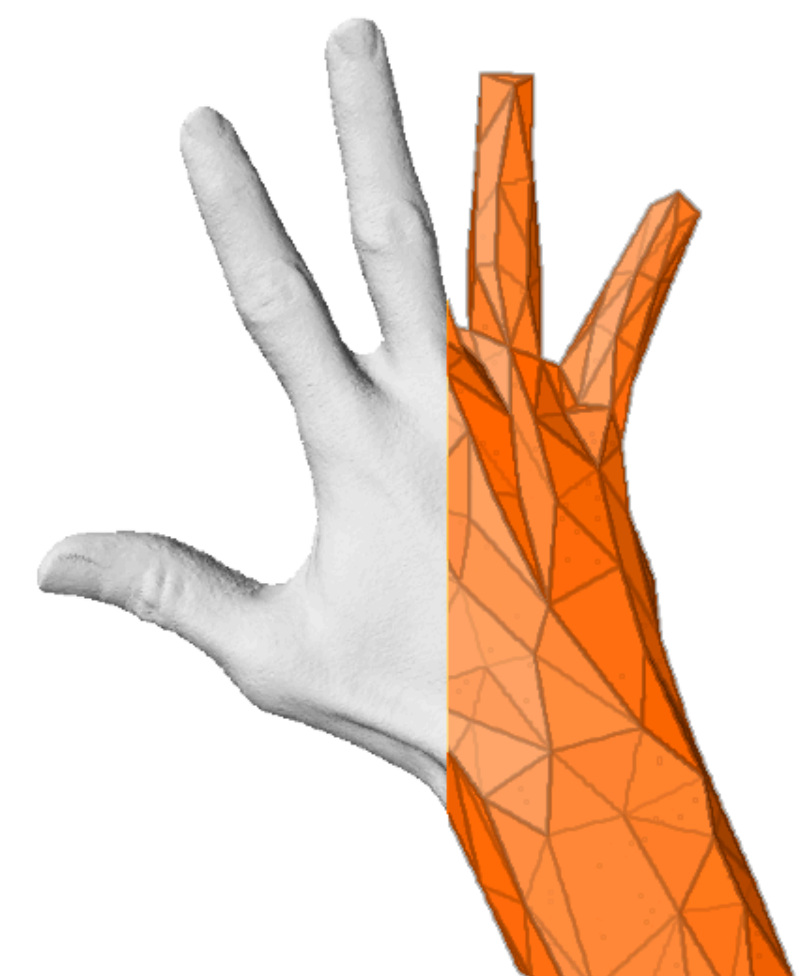


remeshing



shape analysis

parameterization



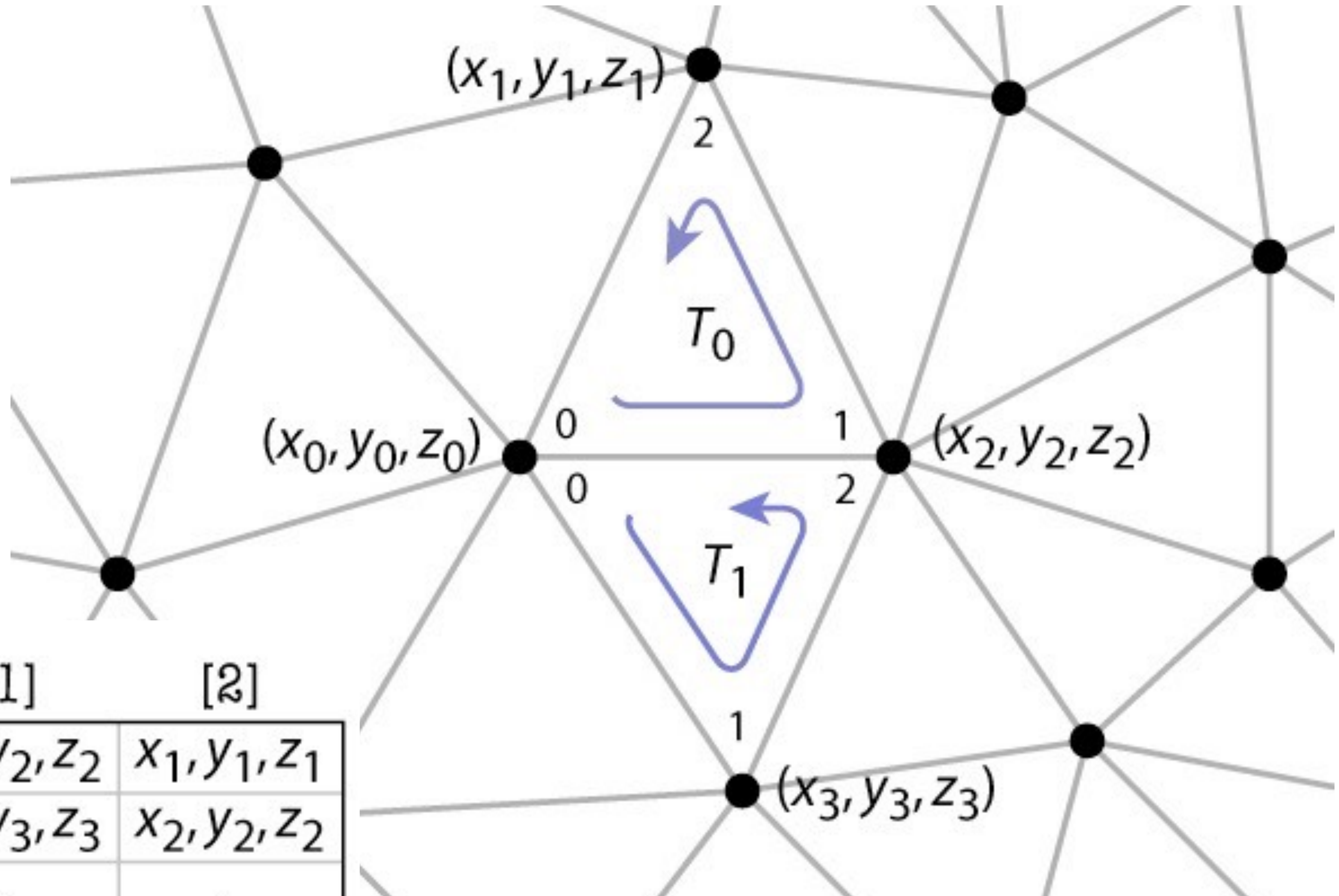
compression

Today

- Study how to represent meshes (data structures)
- Study how to process meshes (basic geometry processing)
    - Subdivision
    - Mesh simplification
    - Mesh resampling

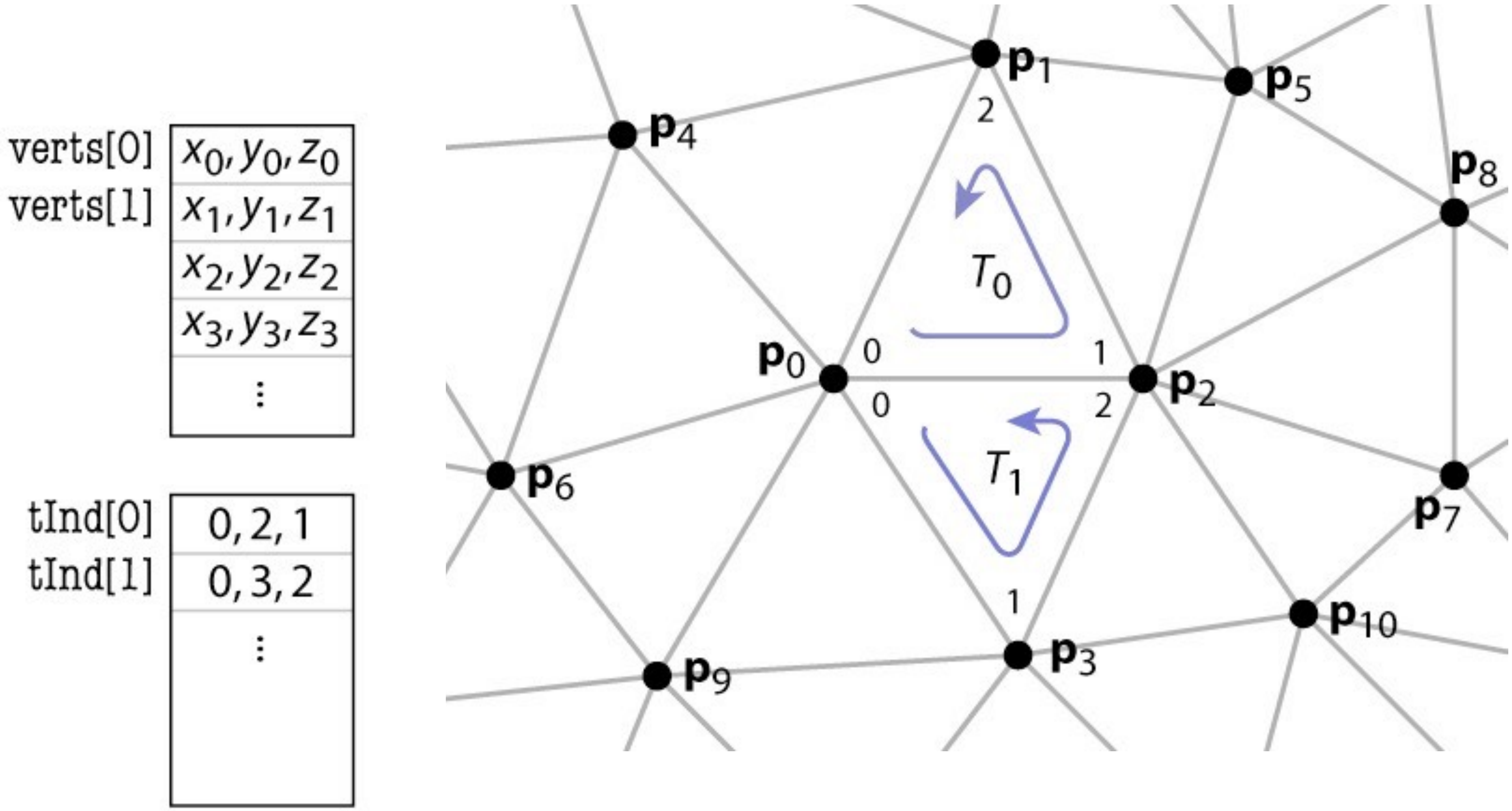
Mesh representations

List of triangles



	[0]	[1]	[2]
tris[0]	x_0, y_0, z_0	x_2, y_2, z_2	x_1, y_1, z_1
tris[1]	x_0, y_0, z_0	x_3, y_3, z_3	x_2, y_2, z_2
	⋮	⋮	⋮

Lists of vertexes / indexed triangle



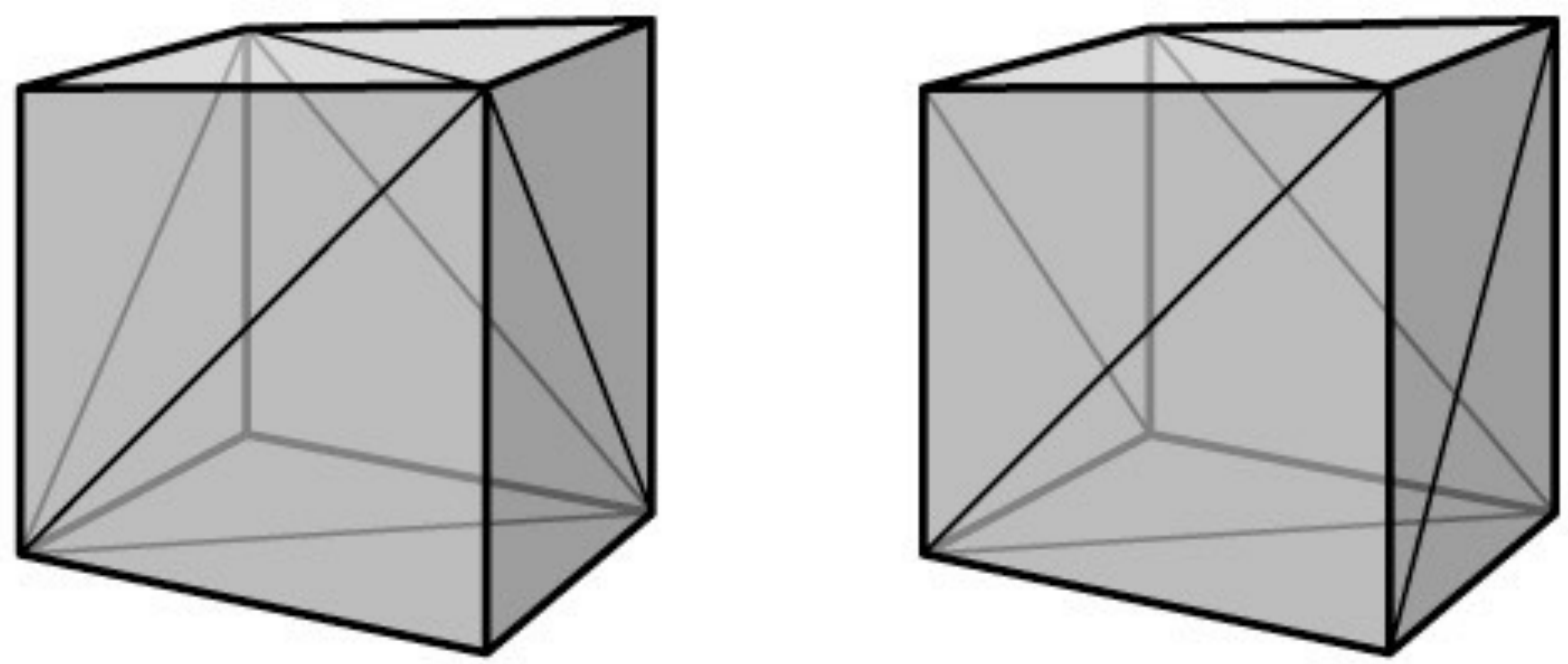
Comparison

- **List of triangles**
  + Simple
  – Contains redundant vertex information

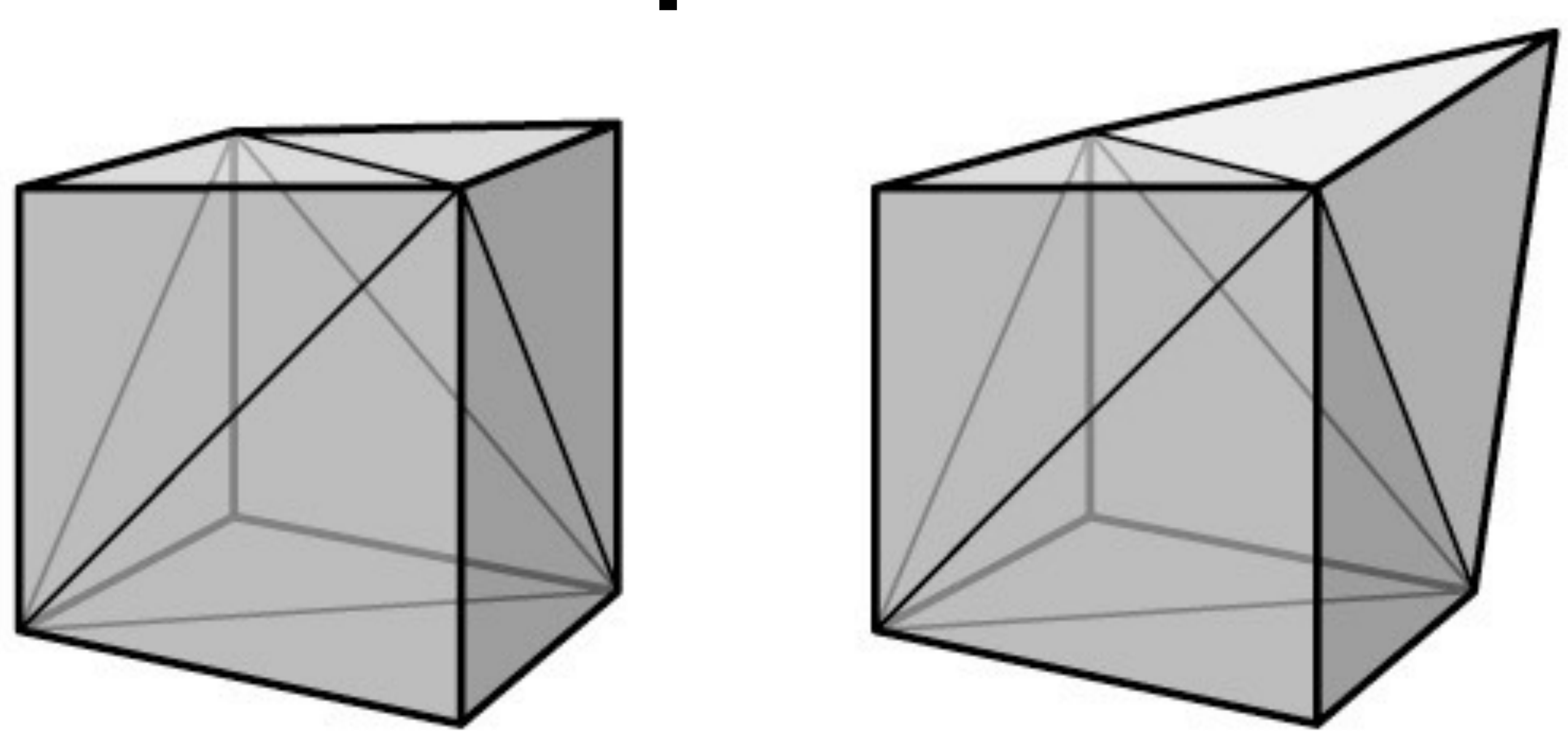
- **Vertexes + indexed triangles**
  + Sharing vertices reduces memory usage
  + Ensure integrity of the mesh (moving a vertex causes that vertex in all the polygons to move)

Mesh topology vs surface geometry

Same vertex positions, different mesh topology



Same topology, different vertex positions



Topological mesh information

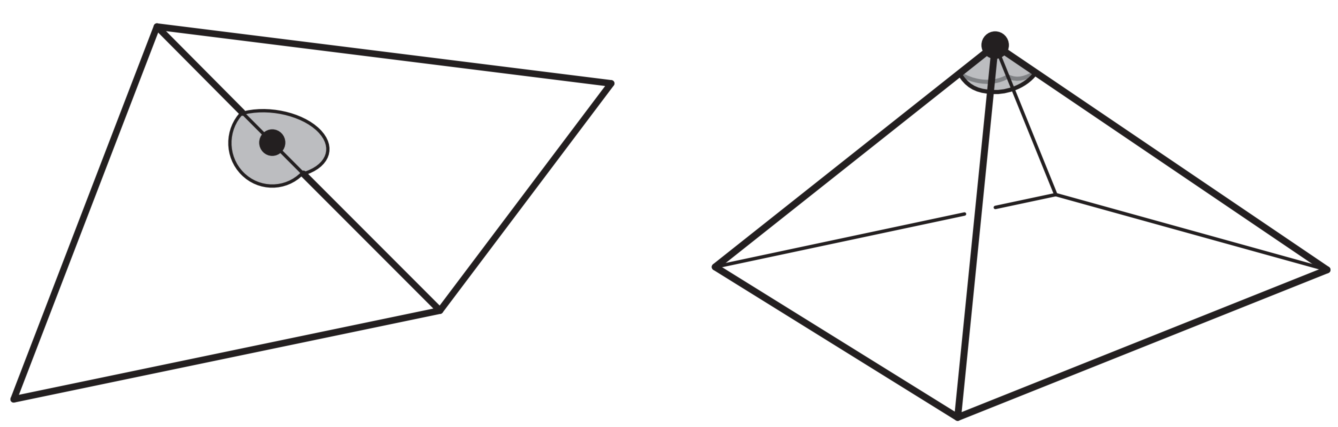
- Applications:
  - Constant time access to neighbors
    e.g. surface normal calculation, subdivision
  - Editing the geometry
    e.g. adding/removing vertices, faces, edges, etc.

- Solution: topological data structures

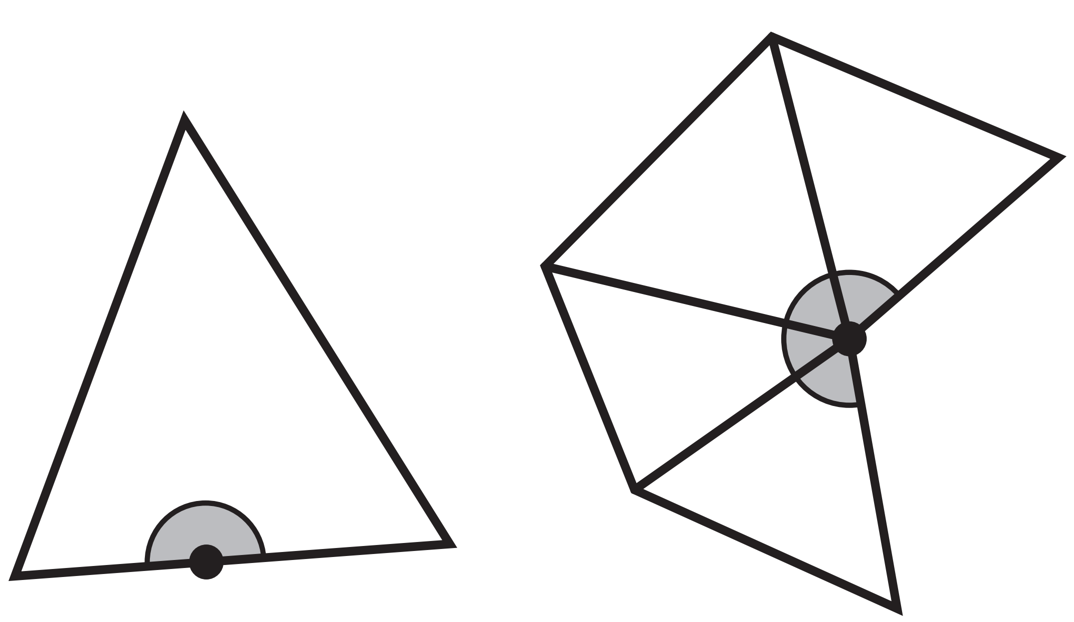
Topological validity: manifold

- Recall, a 2D manifold is a surface that when cut with a small sphere always yields a disk

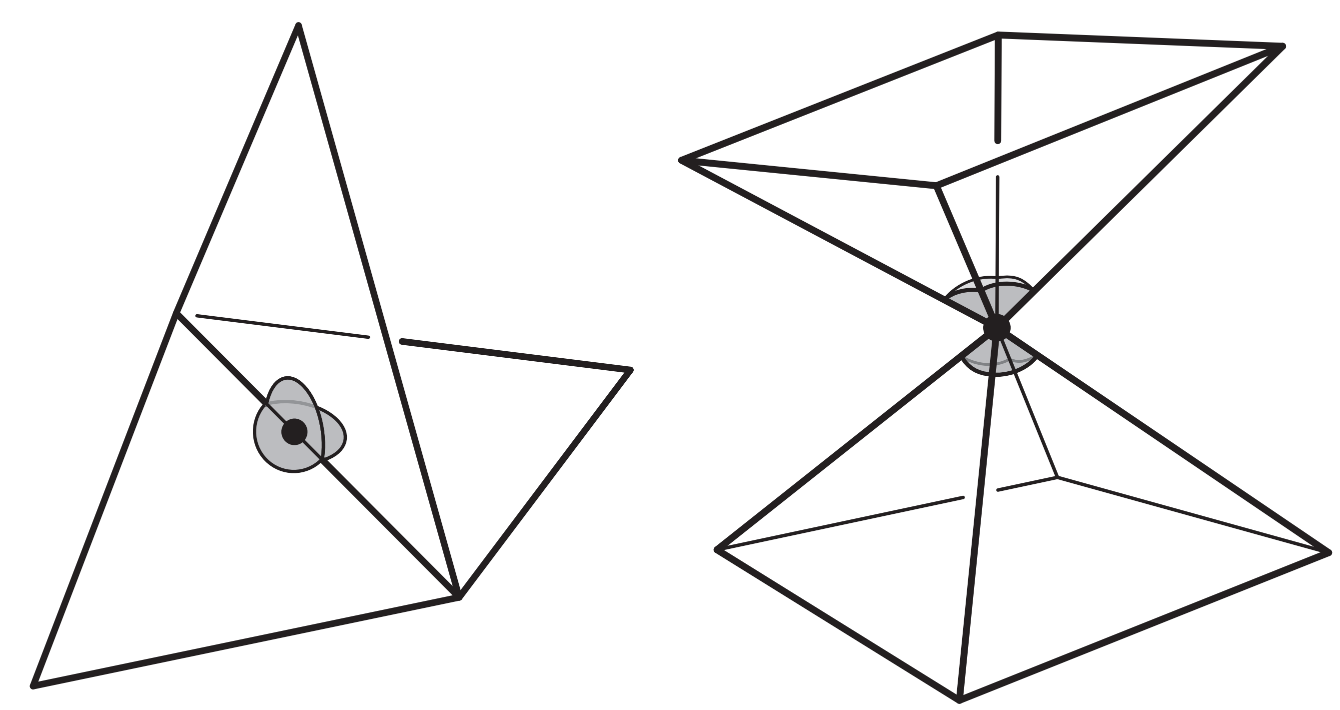
Manifold



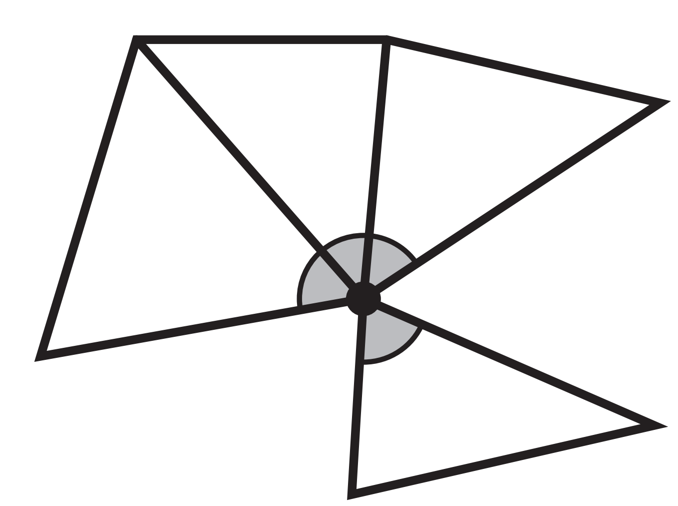
With border



Not manifold



With border



Manifolds have useful properties

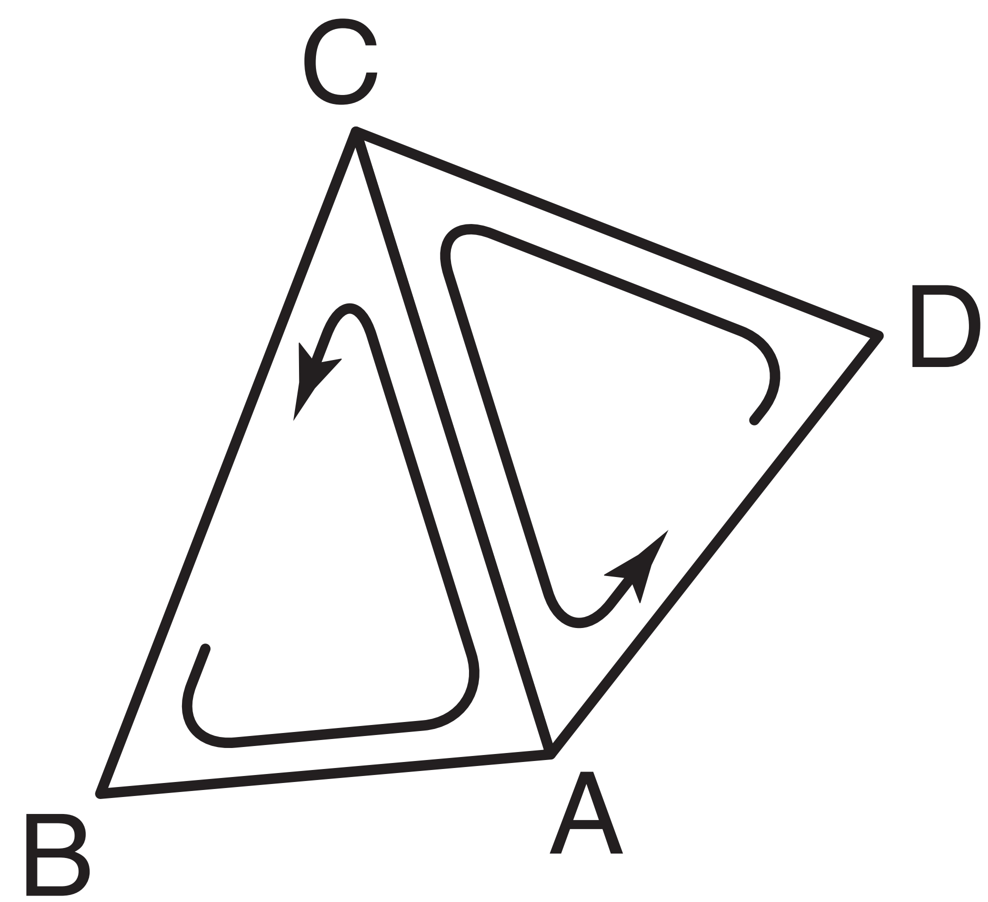
- A 2D manifold is a surface that when cut with a small sphere always yields a disk

- If a mesh is manifold, we can rely on these useful properties: *
  - An edge connects exactly two faces
  - An edge connects exactly two vertices
  - A face consists of a ring of edges and vertices
  - A vertex consists of a ring of edges and faces
  - Euler's polyhedron formula holds: $\#f - \#e + \#v = 2$
    (for a surface topologically equivalent to a sphere)
    (Check for a cube: $6 - 12 + 8 = 2$)

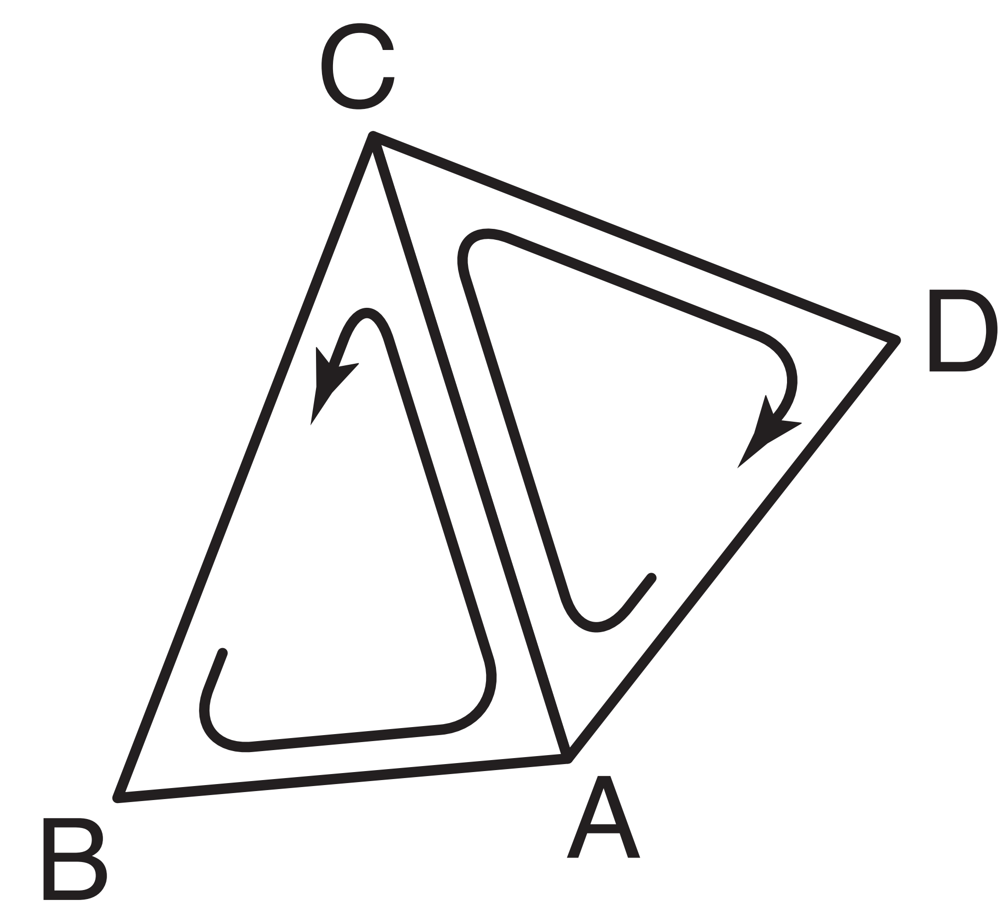
* Some of these properties only apply to non-border mesh regions

Topological validity: orientation consistency

Both facing front



Inconsistent orientations



Non-orientable
(e.g., Moebius strip)

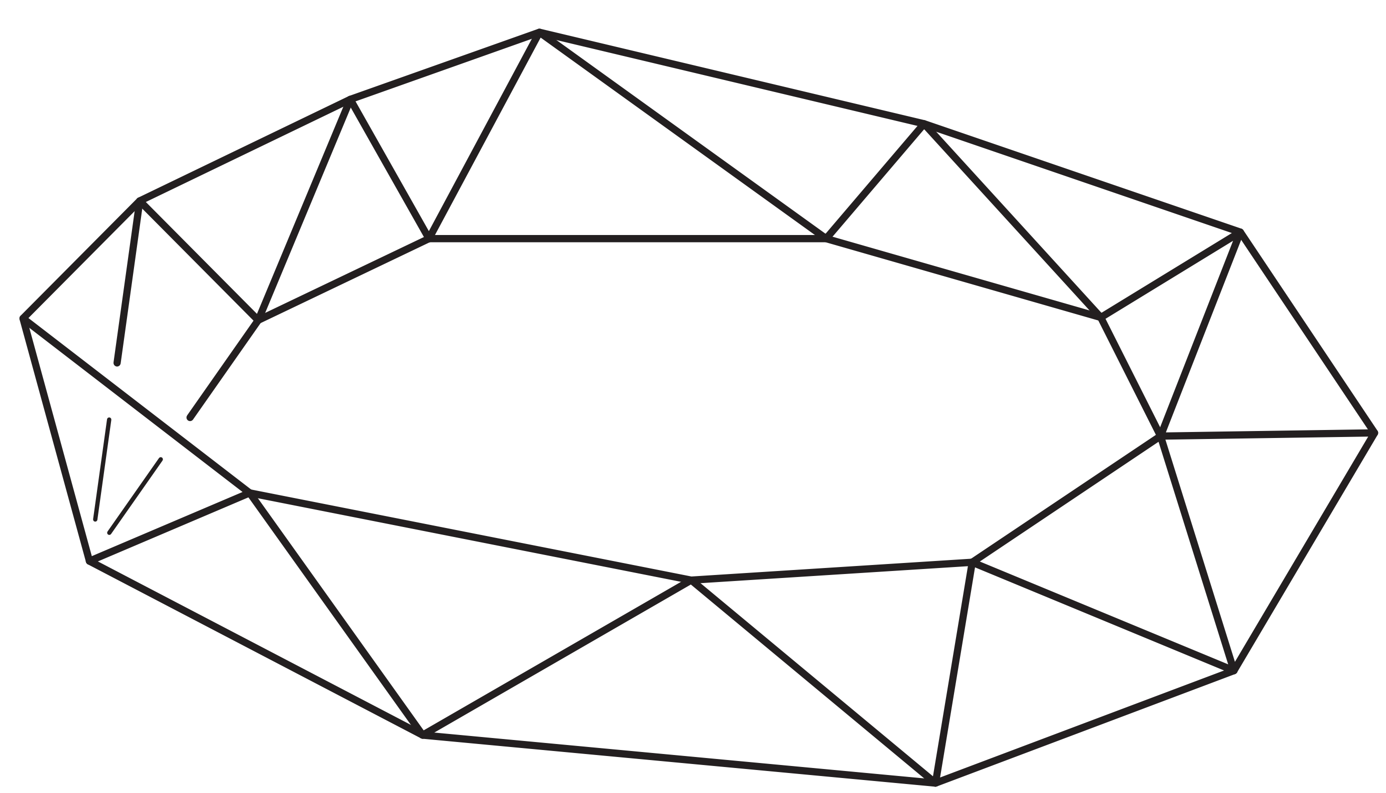
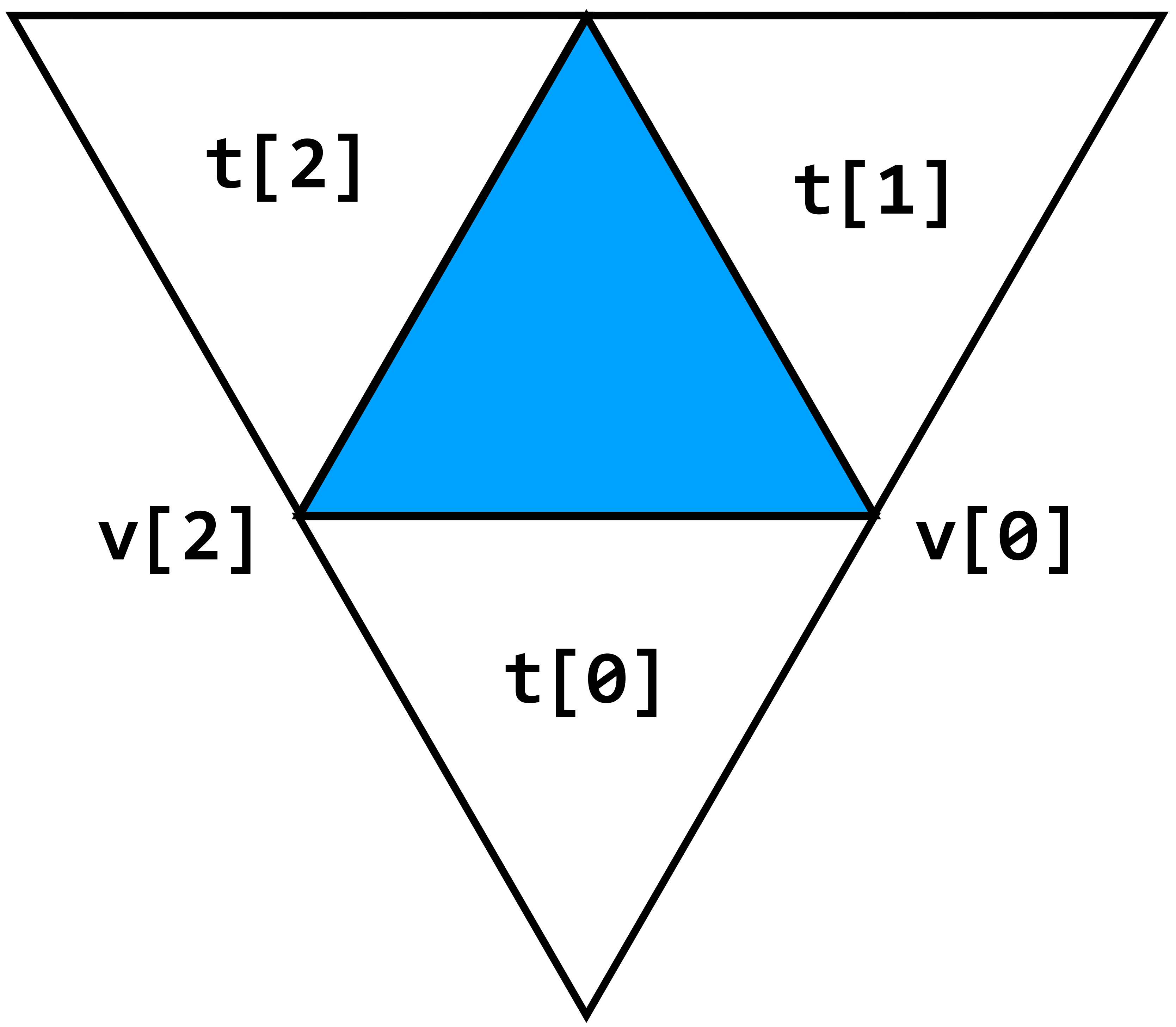


Image credit: Wikipedia

Simple example: triangle-neighbor data structure

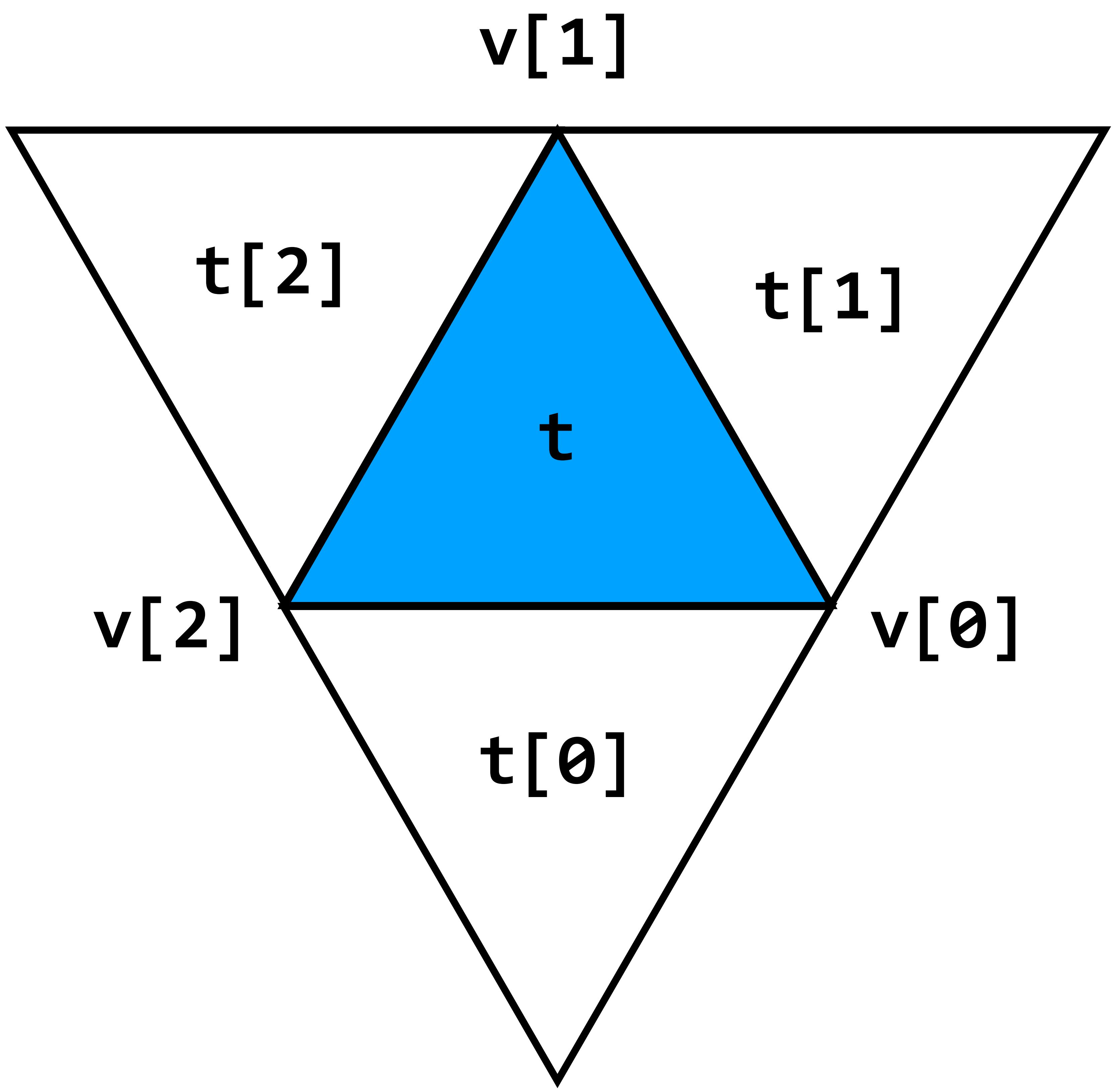
```
struct Tri {
   Vert   * v[3];
   Tri    * t[3];
}

struct Vert {
   Point  pt;
   Tri    *t;
}
```



Triangle-neighbor – mesh traversal

Find next triangle counter-clockwise around vertex v from triangle t

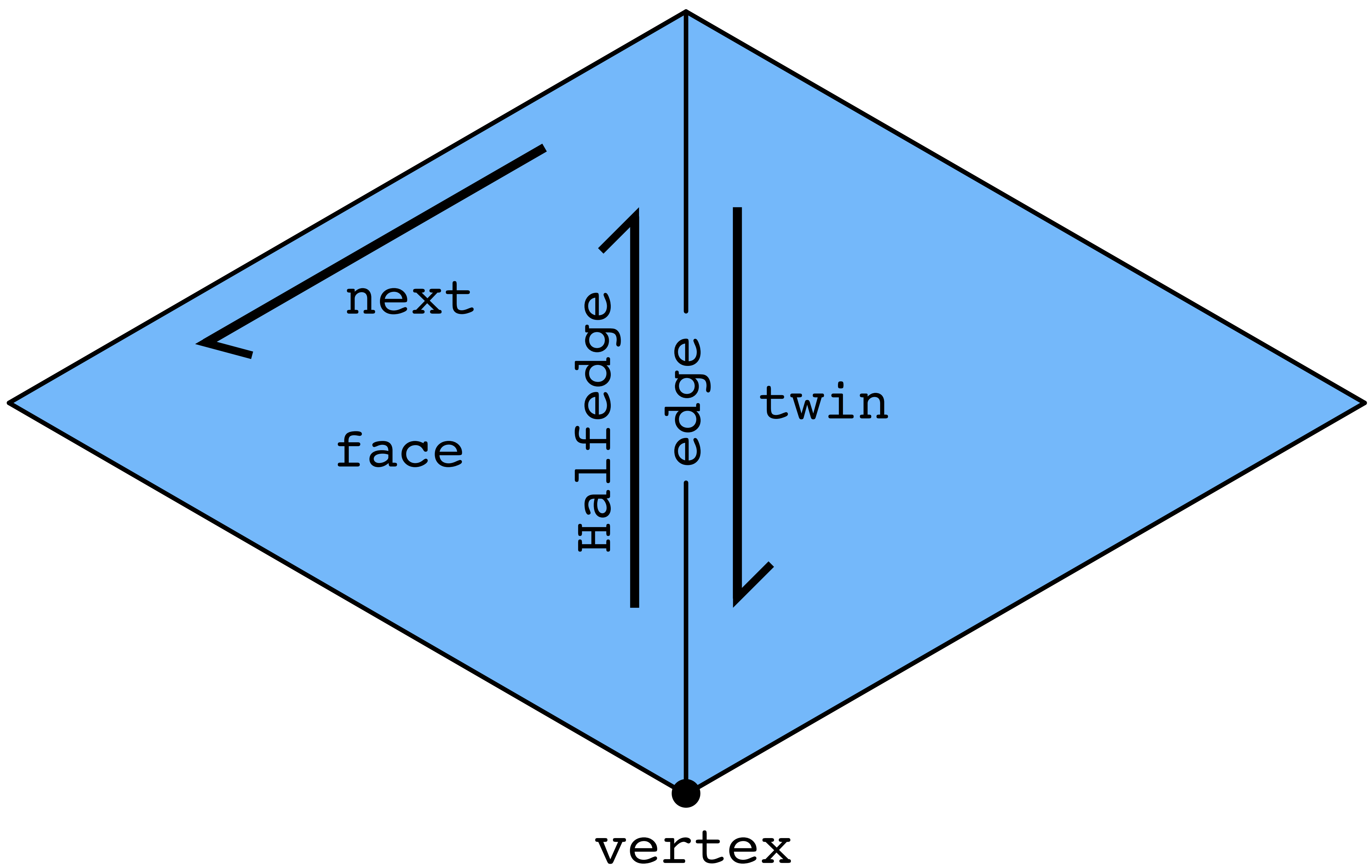
```
Tri* tccwvt(Vert *v, Tri *t)
{
   if (v == t->v[0])
      return t[0];
   if (v == t->v[1])
      return t[1];
   if (v == t->v[2])
      return t[2];
}
```



Half-edge data structure

```
struct Halfedge {
   Halfedge *twin,
   Halfedge *next;
   Vertex *vertex;
   Edge *edge;
   Face *face;
}
struct Vertex {
   Point pt;
   Halfedge *halfedge;
}
struct Edge {
   Halfedge *halfedge;
}
struct Face {
   Halfedge *halfedge;
}
```

Key idea: two half-edges act as "glue" between mesh elements



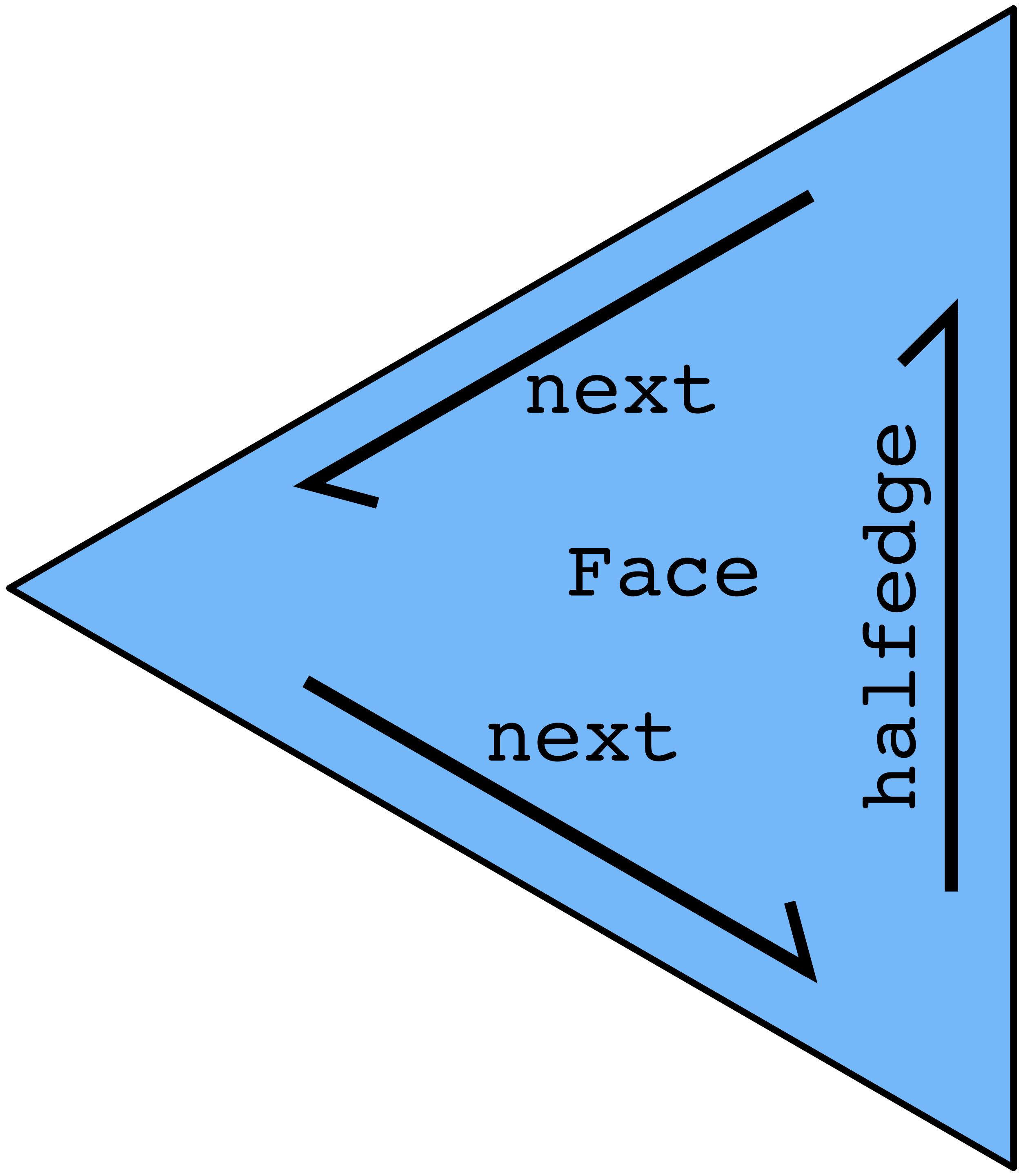
Each vertex, edge and face points to one of its half edges

Half-edge structure facilitates mesh traversal

- Use twin and next pointers to move around mesh
- Process vertex, edge and/or face pointers

Example 1: process all vertices of a face

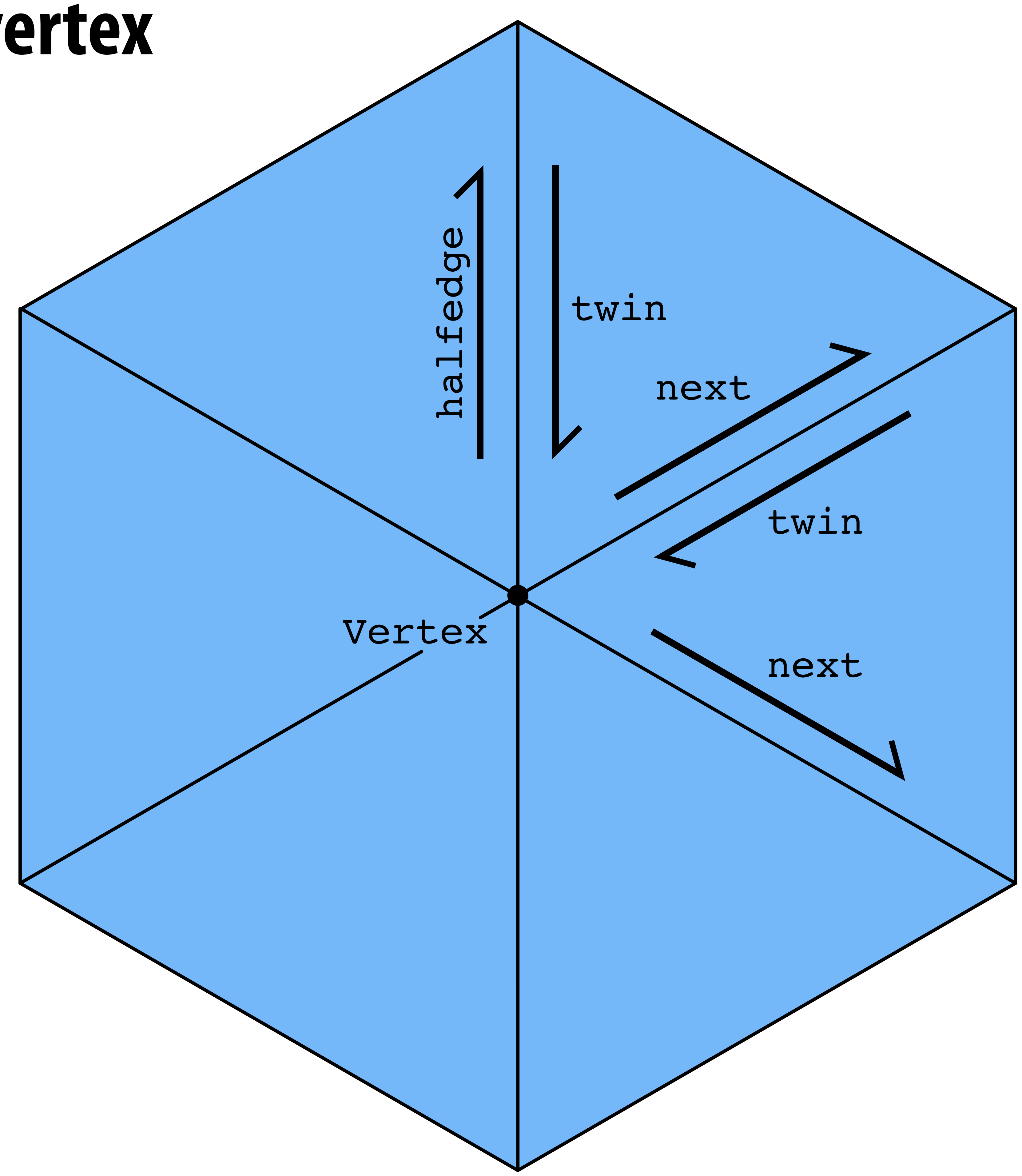
```
Halfedge* h = f->halfedge;
do {
   process(h->vertex);
   h = h->next;
}
while( h != f->halfedge );
```



Half-edge structure facilitates mesh traversal

Example 2: process all edges around a vertex

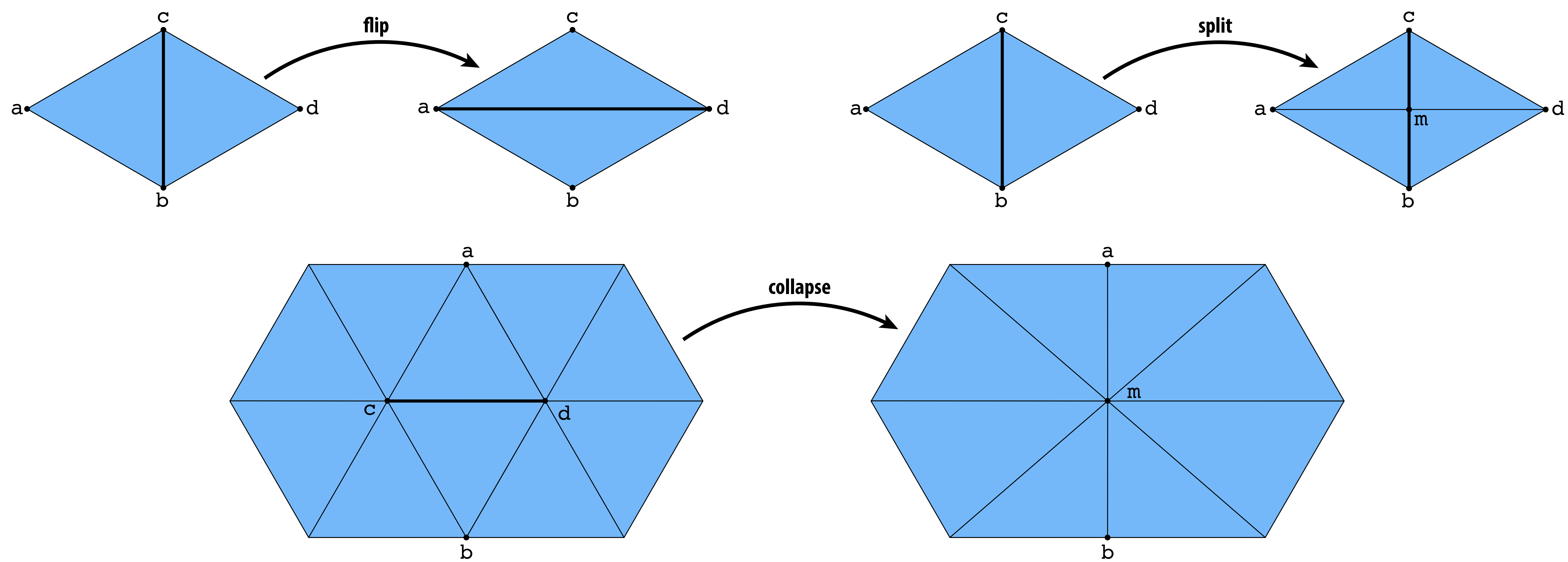
```
Halfedge* h = v->halfedge;
do {
   process(h->edge);
   h = h->twin->next;
}
while( h != v->halfedge );
```



Local mesh operations

Half-Edge – local mesh editing

- Consider basic operations for linked list: insert, delete
- Basic ops for half-edge mesh: flip, split, collapse edges

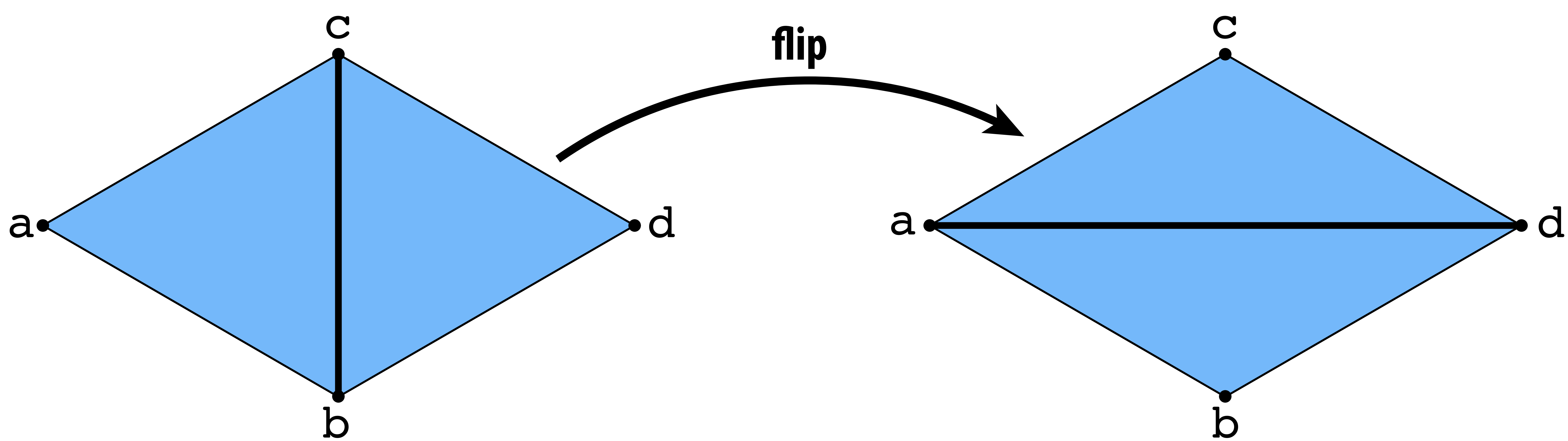


Allocate / delete elements; reassign pointers

(Care is needed to preserve mesh manifold property)

Half-edge – edge flip

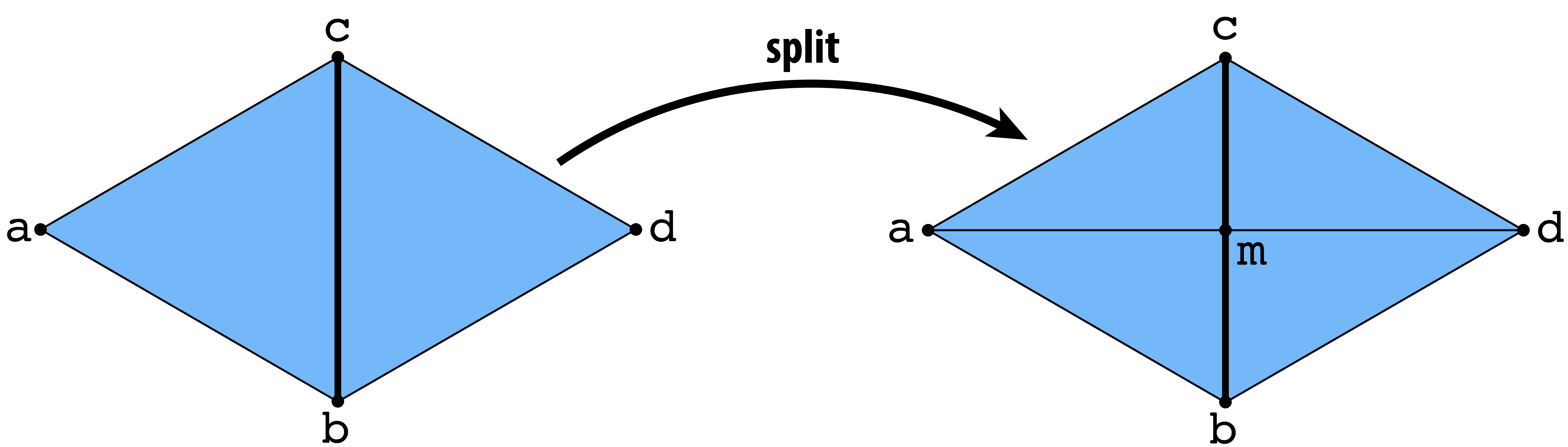
Triangles (a,b,c), (b,d,c) become (a,d,c), (a,b,d):



- Long list of half-edge pointer reassignments
- However, no mesh elements created/destroyed

Half-edge – edge split

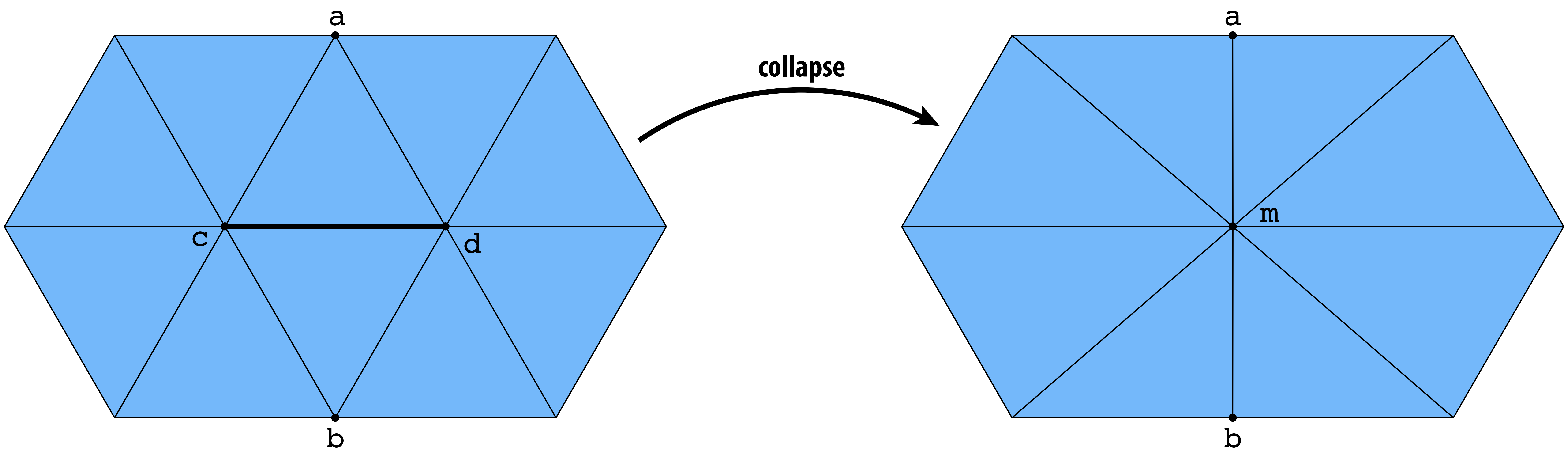
Insert midpoint m of edge (c,b), connect to get four triangles:



- **Must add elements to mesh (new vertex, faces, edges)**
- **Again, many half-edge pointer reassignments**

Half-edge – edge collapse

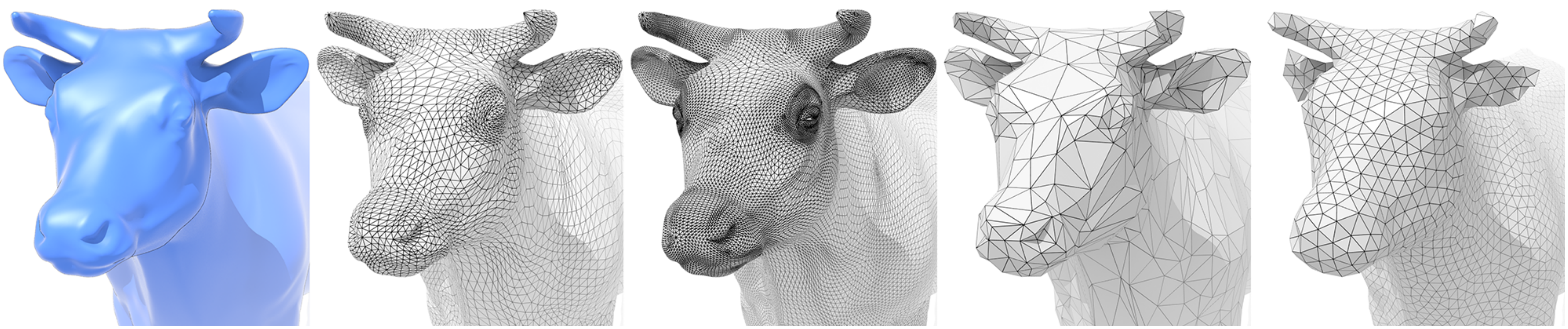
Replace edge (c,d) with a single vertex m:



- Must delete elements from the mesh
- Again, many half-edge pointer reassignments

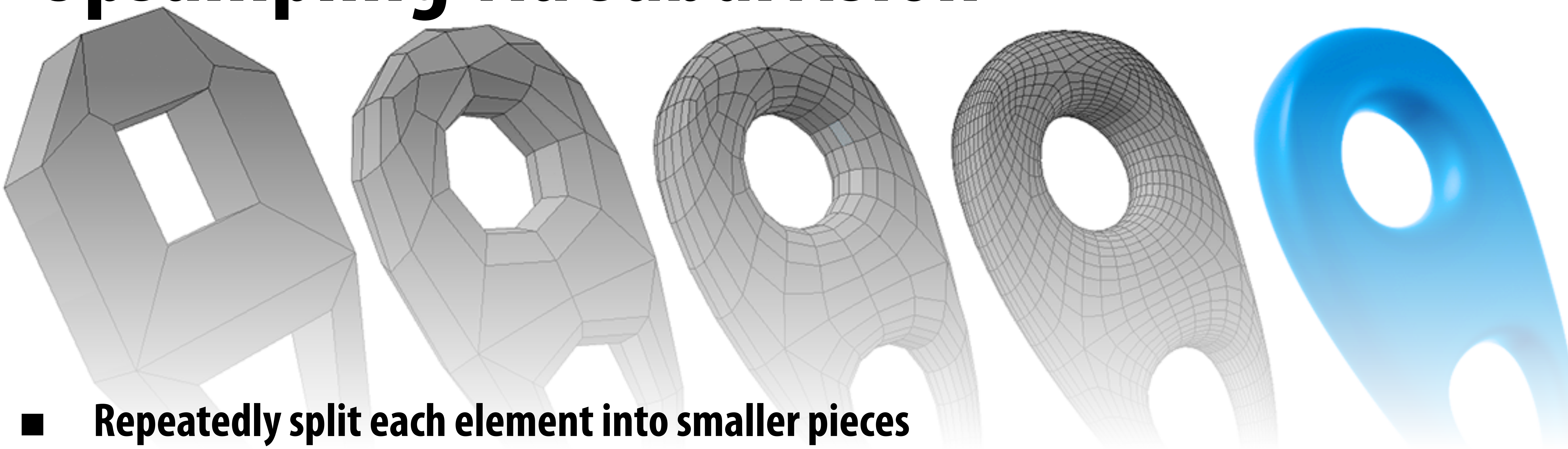
Global mesh operations: geometry processing

- Mesh subdivision (form of subsampling)
- Mesh simplification (form of downsampling)
- Mesh regularization (form of resampling)

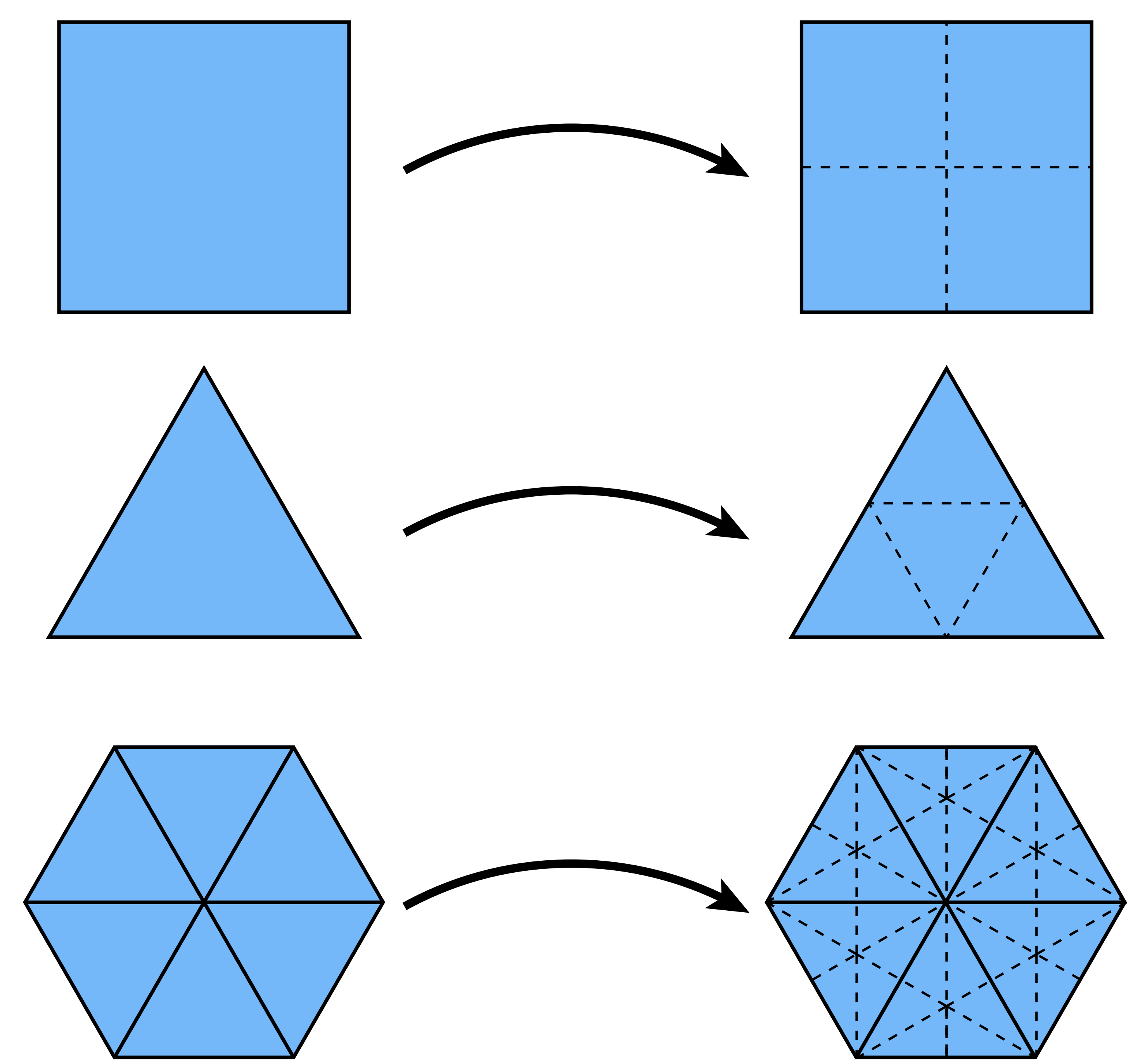


Upsampling a mesh — subdivision

Upsampling via subdivision

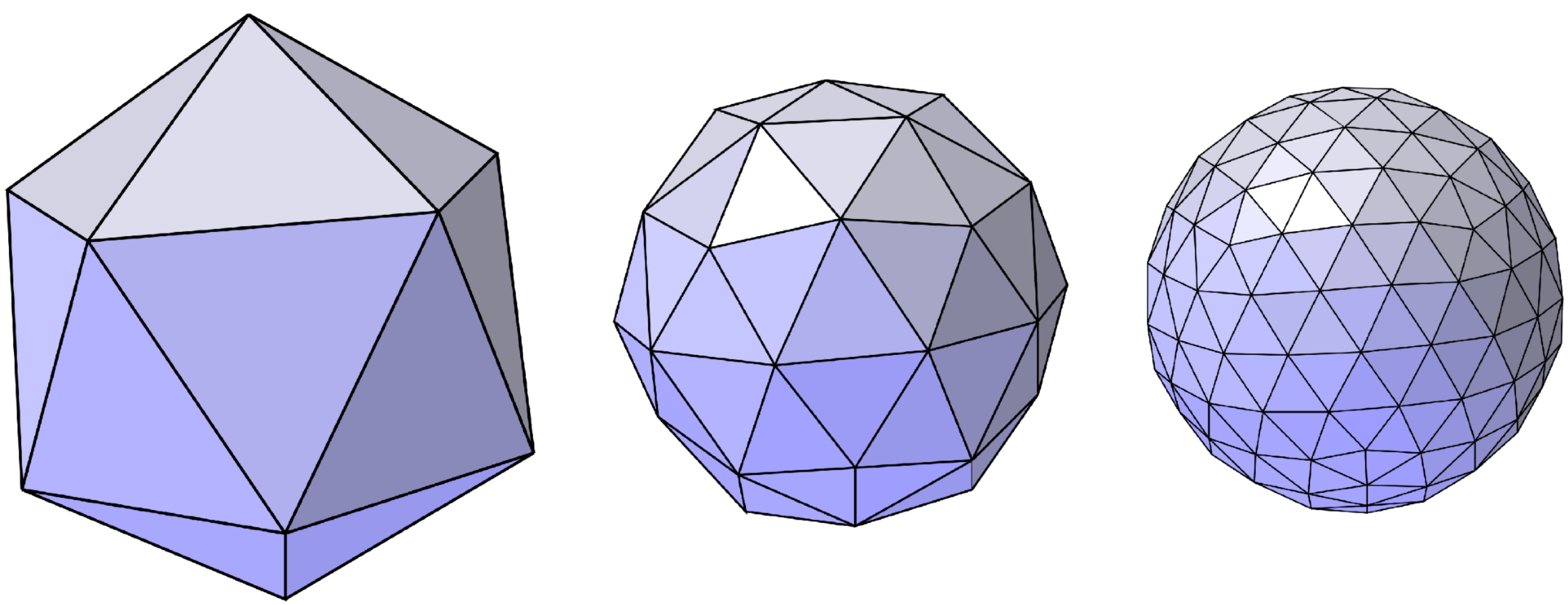


- Repeatedly split each element into smaller pieces
- Replace vertex positions with weighted average of neighbors
- Main considerations:
  - interpolating vs. approximating
  - limit surface continuity (C^1, C^2, ...)
  - behavior at irregular vertices
- Many options:
  - Quad: Catmull-Clark
  - Triangle: Loop, butterfly, sqrt(3)



Loop subdivision

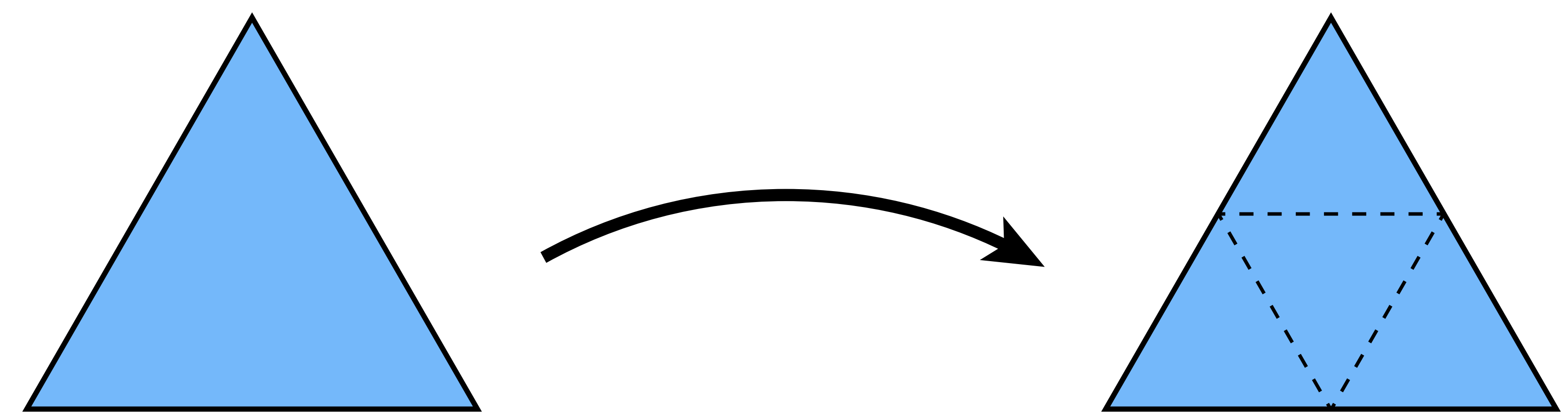
- **Common subdivision rule for triangle meshes**
- **“C2” smoothness away from irregular vertices**
- **Approximating, not interpolating**



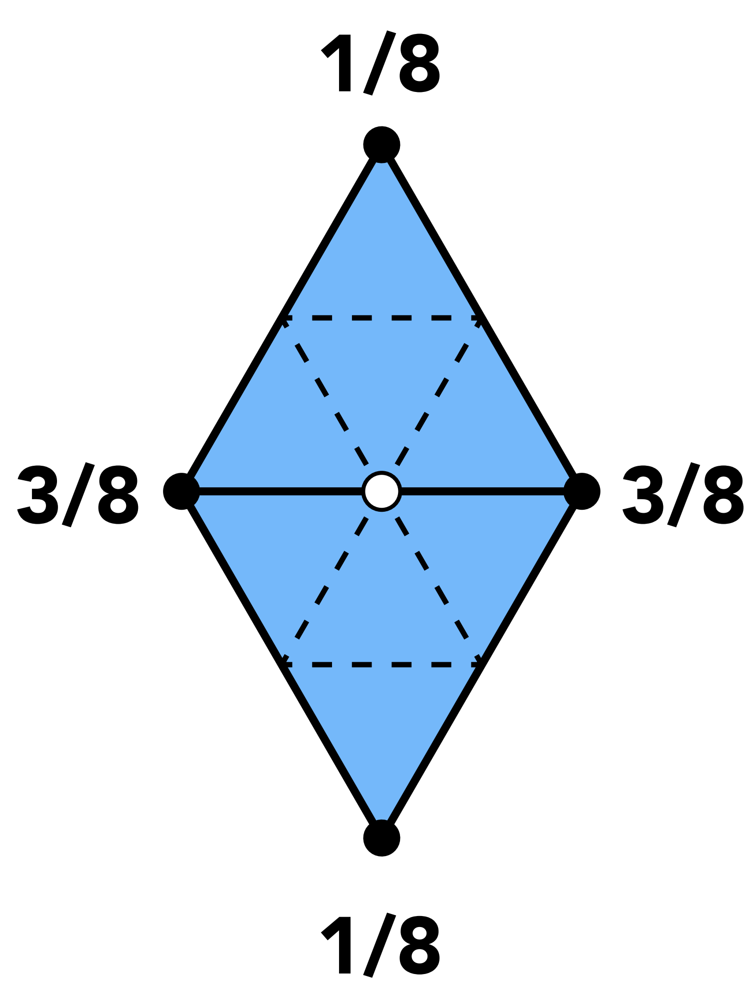
Simon Fuhrman

Loop subdivision algorithm

- Split each triangle into four

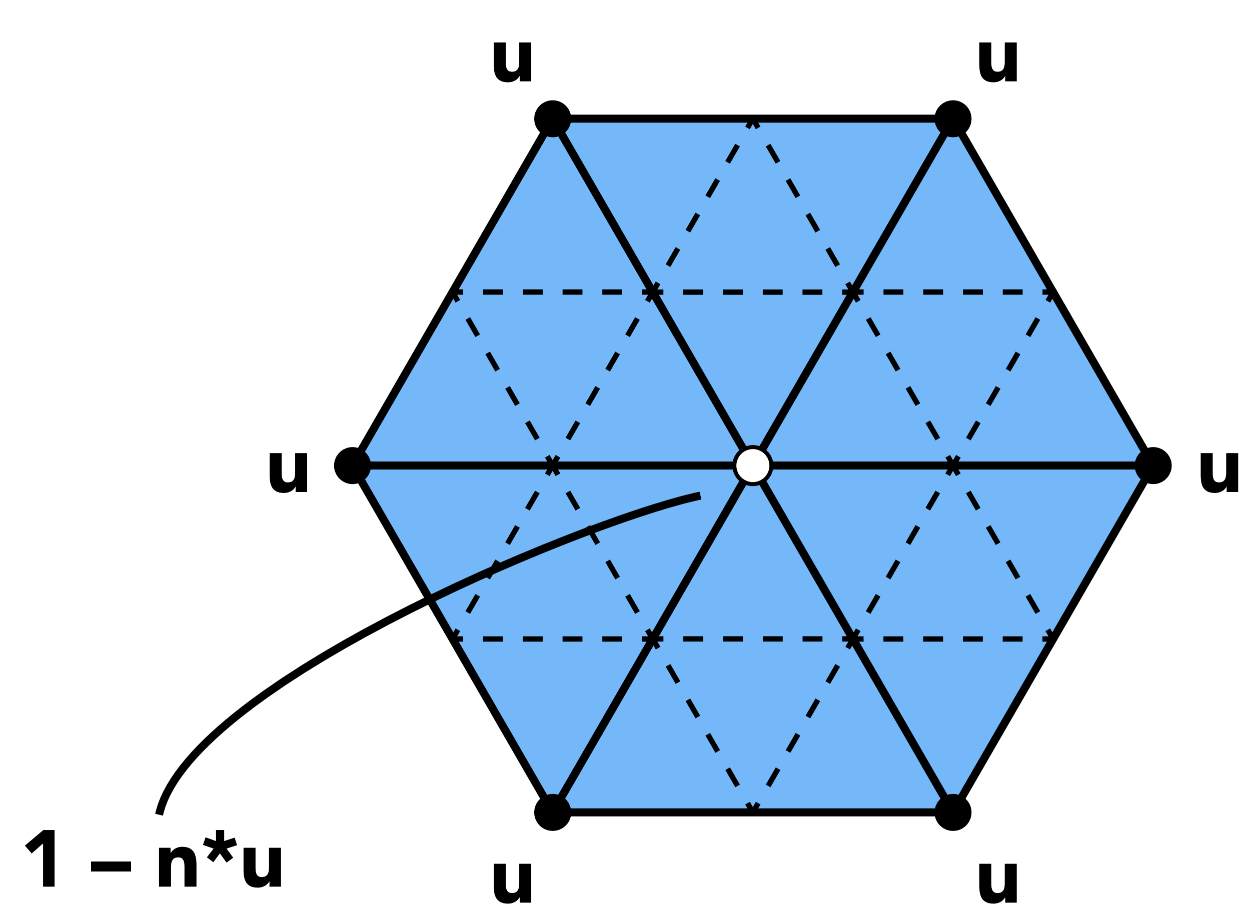


- Compute new vertex positions using weighted sum of prior vertex positions:



New vertices

(weighted sum of vertices on split edge, and vertices "across from" edge)



Old vertices

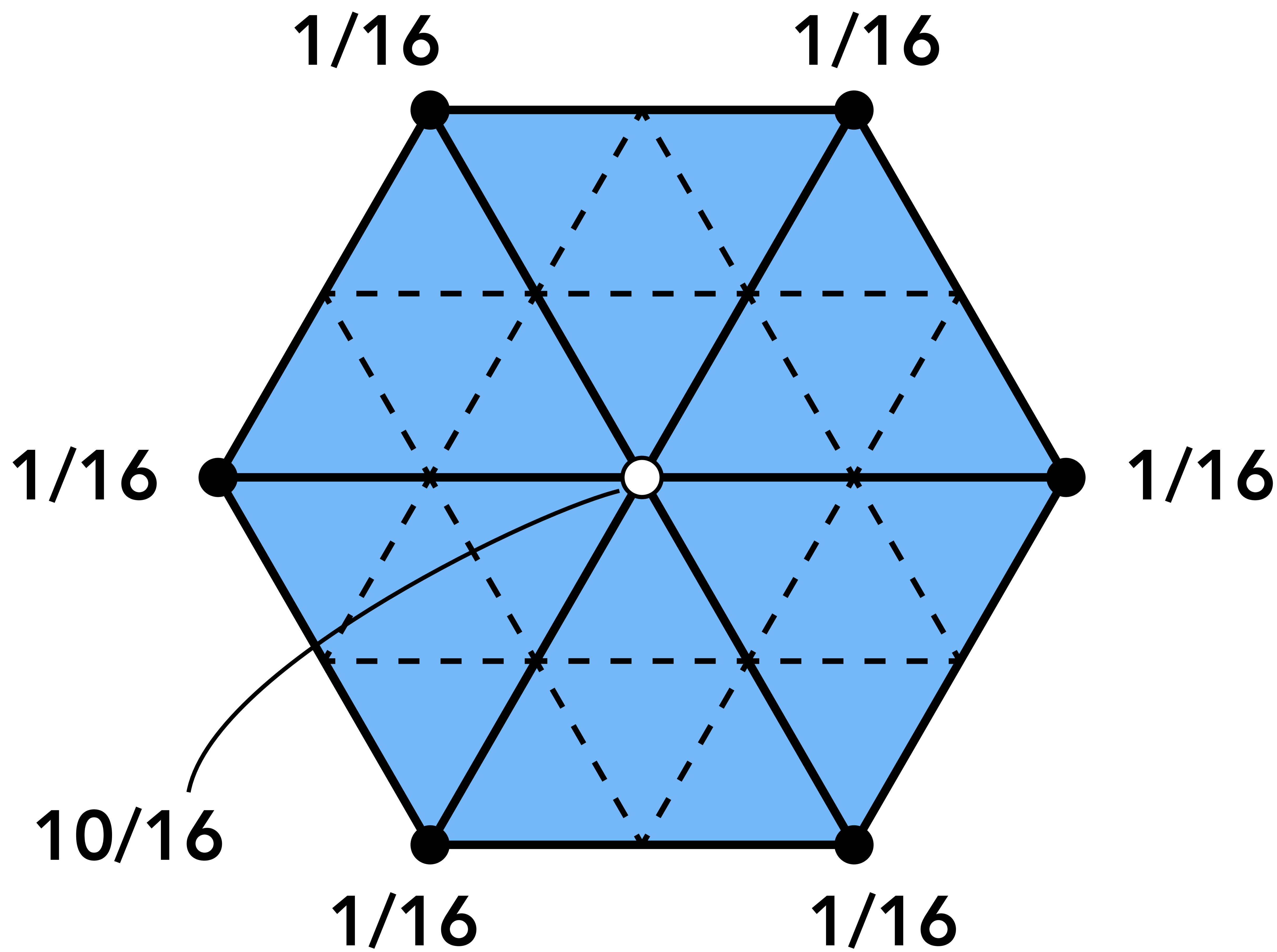
(weighted sum of edge adjacent vertices)

n: vertex degree

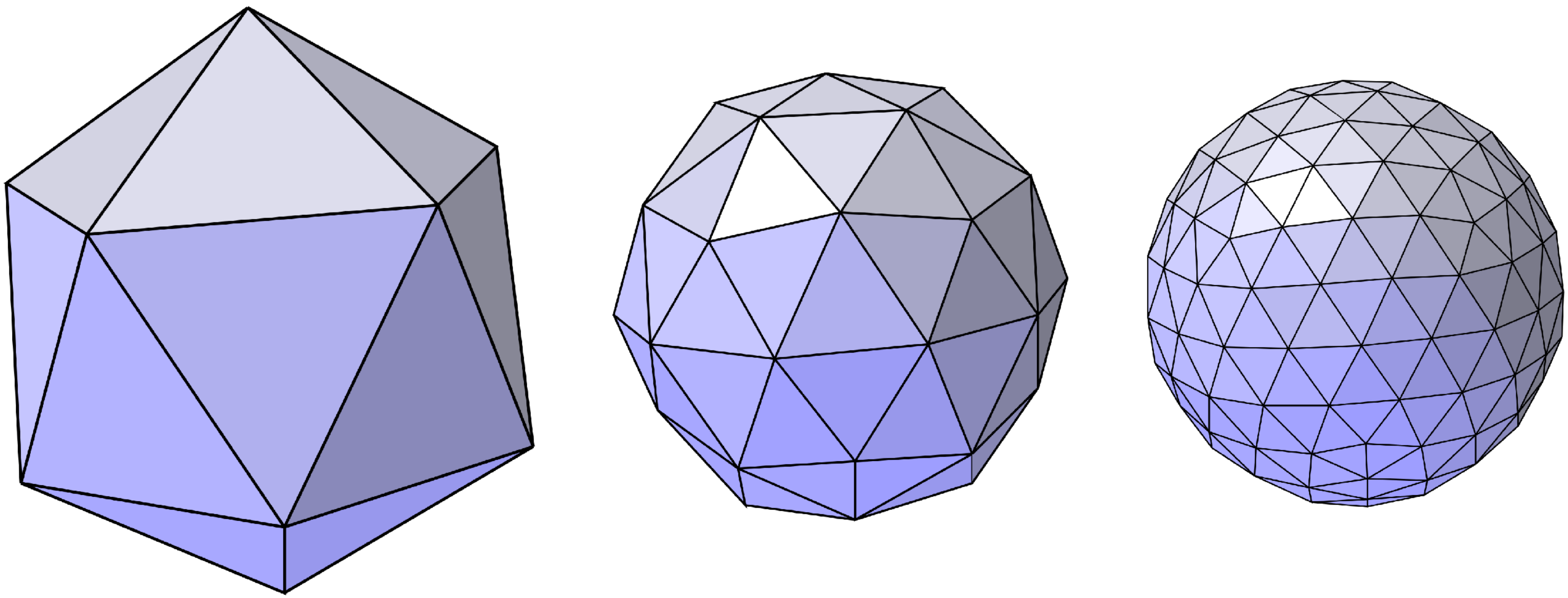
u: 3/16 if n=3, 3/(8n) otherwise

Loop subdivision algorithm

- Example, for degree 6 vertices ("regular" vertices)



Loop subdivision results



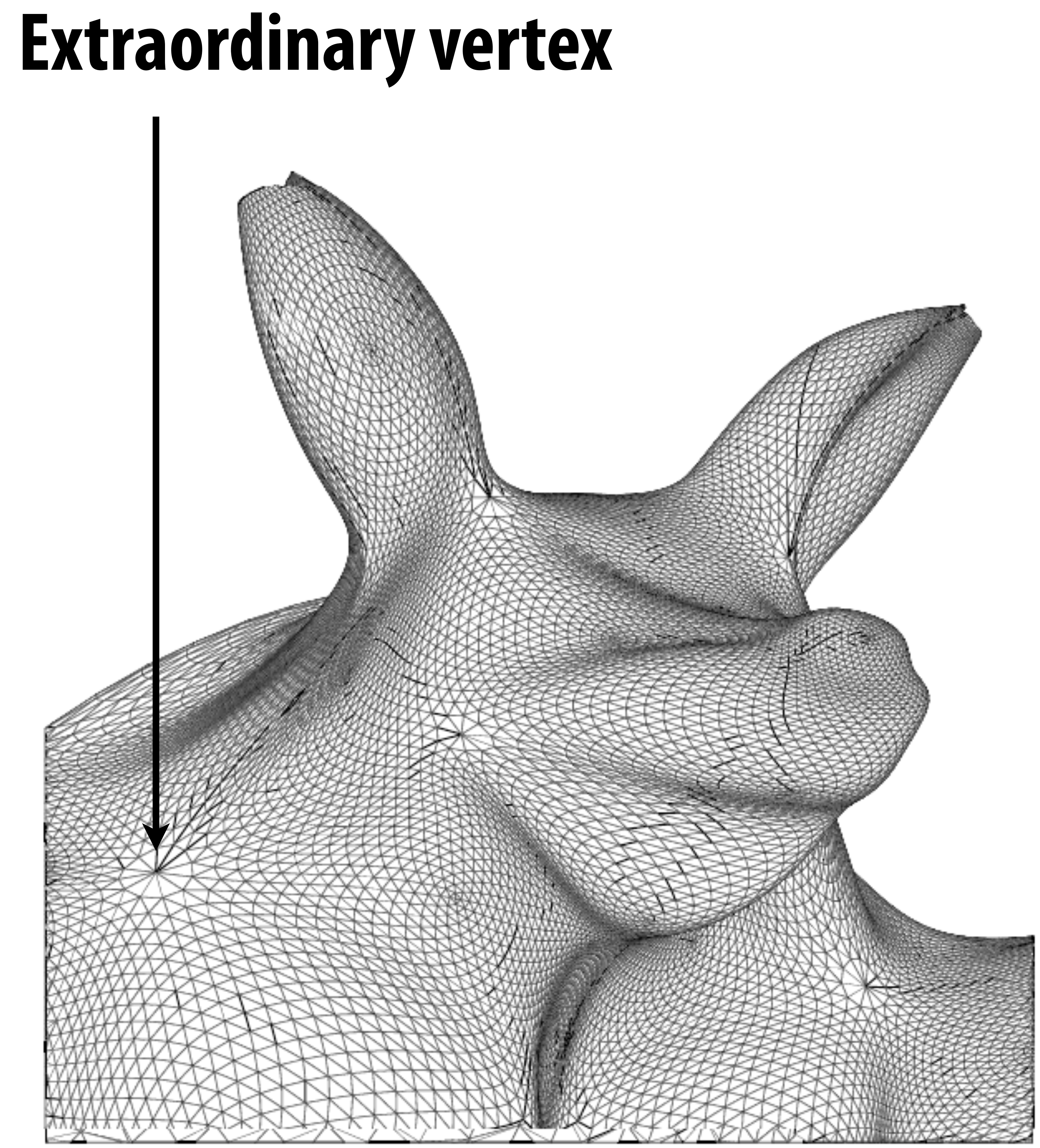
Simon Fuhrman

Semi-regular meshes

Most of the mesh has vertices with degree 6

But if the mesh is topologically equivalent to a sphere, then not all the vertices can have degree 6

Must have a few extraordinary points (degree not equal to 6)



Proof: always an extraordinary vertex

Our triangle mesh (topologically equivalent to sphere) has V vertices, E edges, and T triangles

$E = 3/2\,T$

- There are 3 edges per triangle, and each edge is part of 2 triangles
- Therefore $E = 3/2T$

$T = 2V - 4$

- Euler Convex Polyhedron Formula: $T - E + V = 2$
- => $V = 3/2\,T - T + 2$ => $T = 2V - 4$

If all vertices had 6 triangles, $T = 2V$

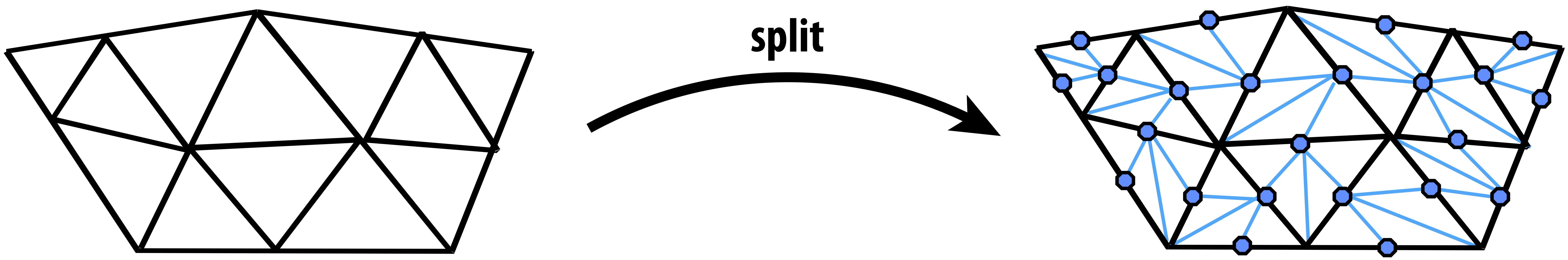
- There are 6 edges per vertex, and every edge connects 2 vertices
- Therefore, $E = 6/2V$ => $3/2T = 6/2V$ => $T = 2V$

T cannot equal both $2V - 4$ and $2V$, a contradiction

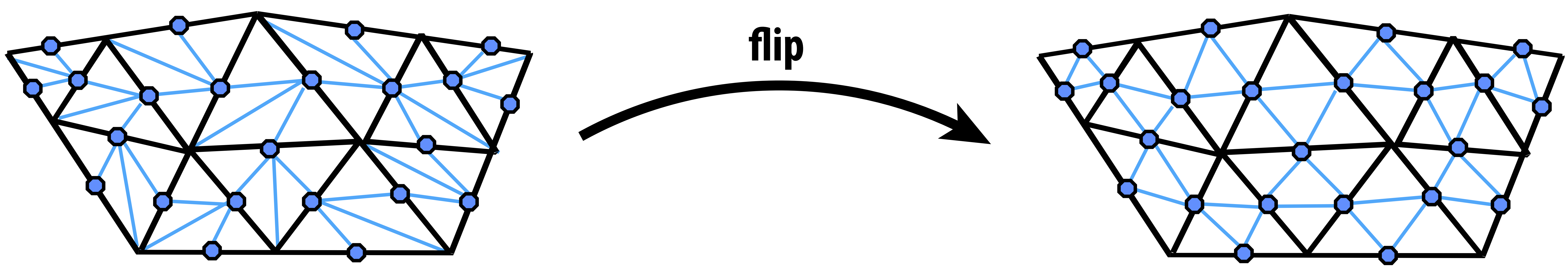
- Therefore, the mesh cannot have 6 triangles for every vertex

Loop subdivision via edge operations

First, split edges of original mesh in any order:



Next, flip new edges that touch a new and old vertex:



(Don't forget to update vertex positions!)

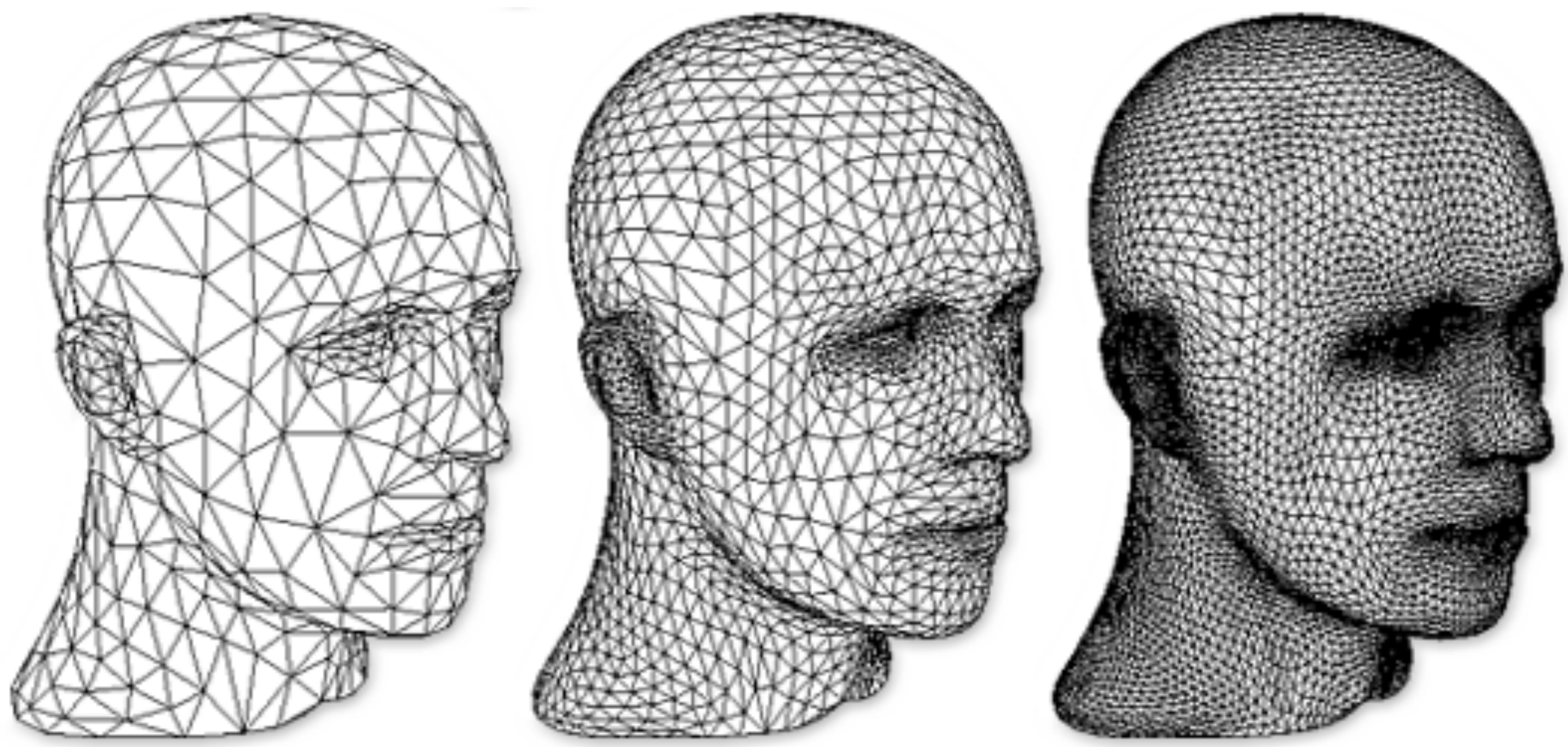
Images cribbed from Keenan Crane, cribbed from Denis Zorin

Continuity of loop subdivision surface

- At extraordinary vertices
  - Surface is at least C^1 continuous

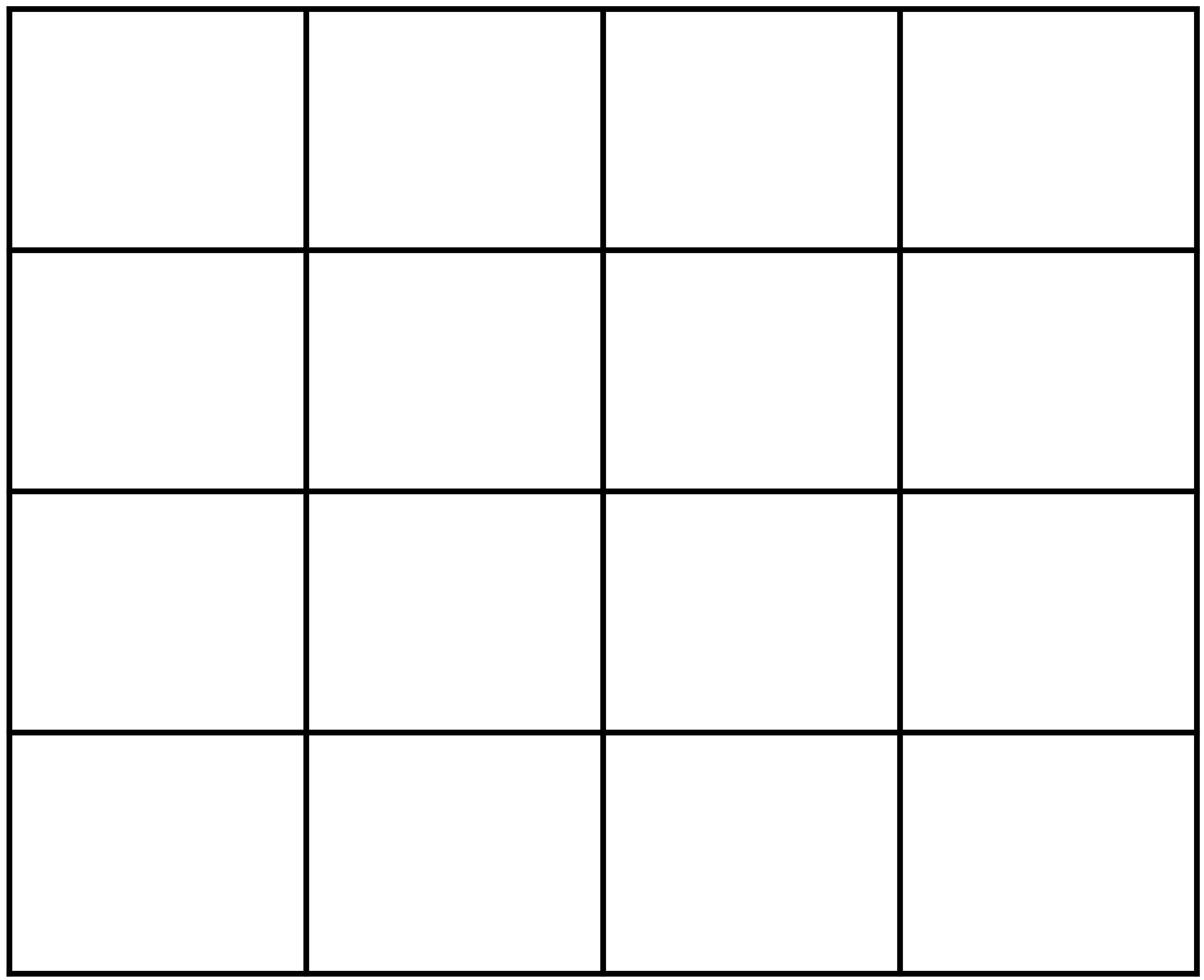
- Everywhere else ("ordinary" regions)
  - Surface is C^2 continuous

Loop subdivision results

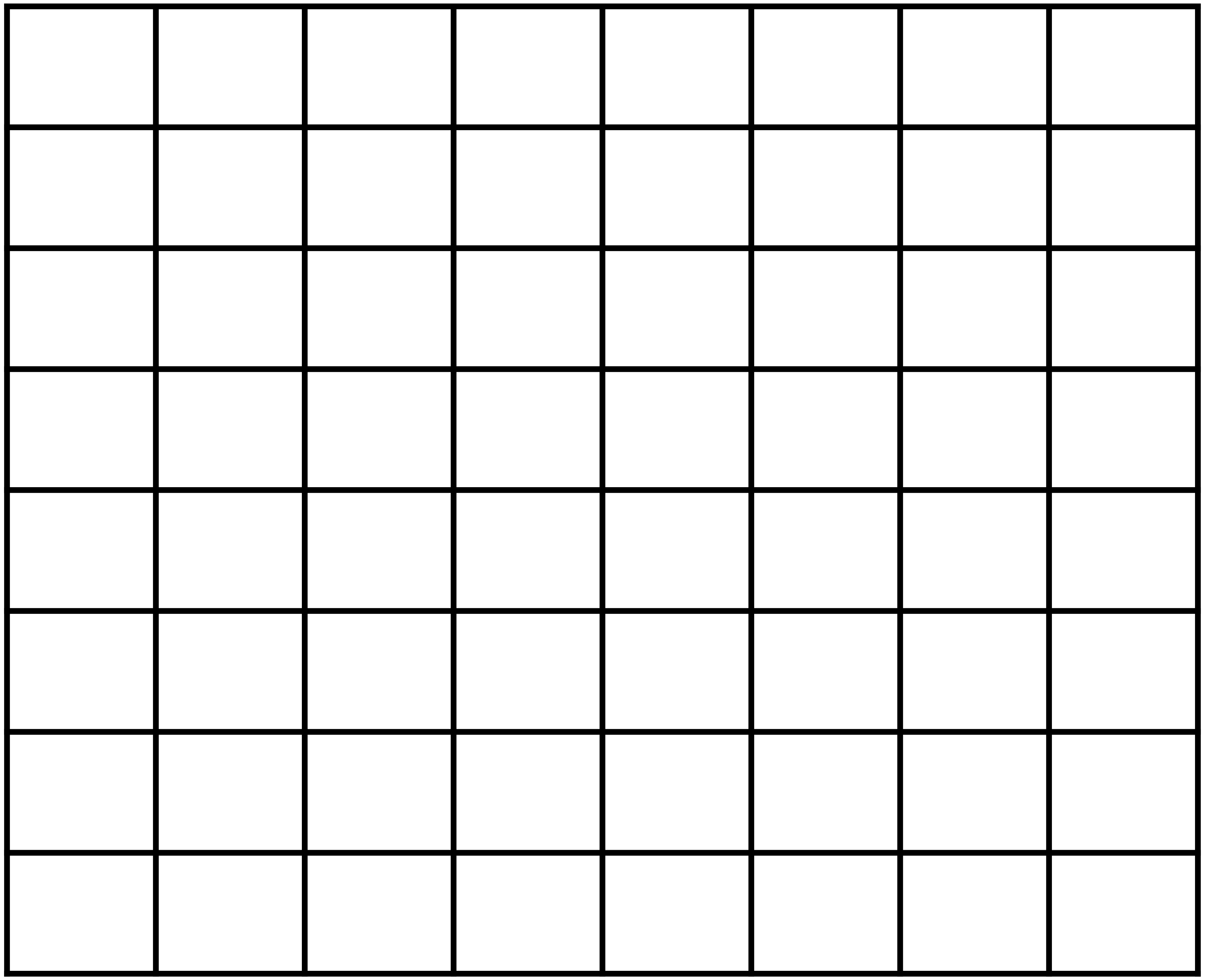


Catmull-Clark subdivision

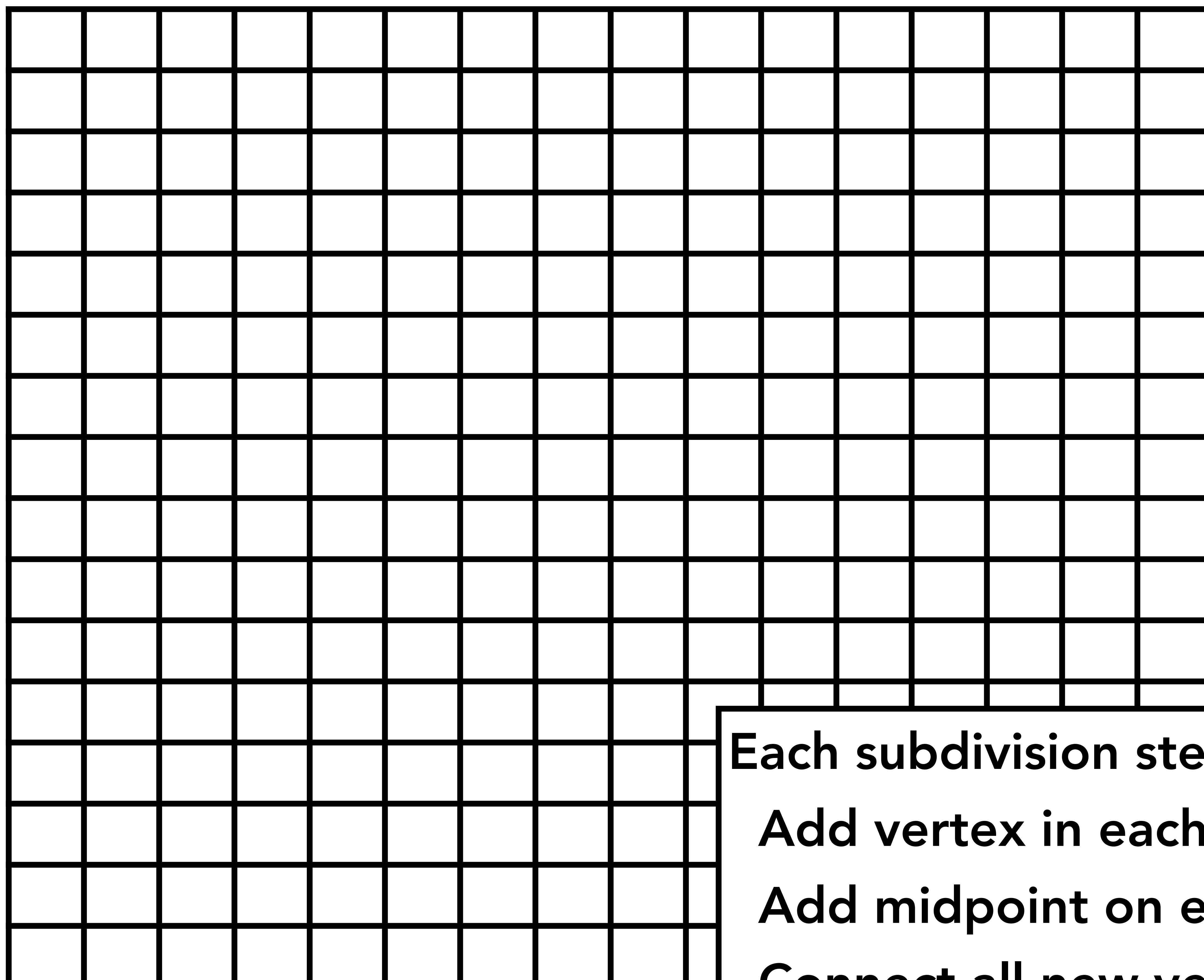
Catmull-Clark subdivision (regular quad mesh)



Catmull-Clark subdivision (regular quad mesh)



Catmull-Clark subdivision (regular quad mesh)

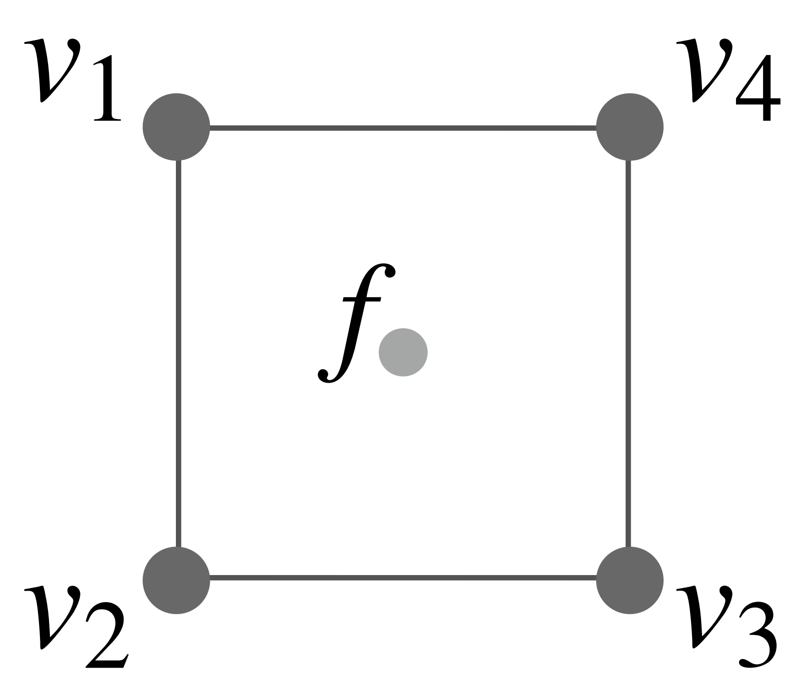


Each subdivision step:

- Add vertex in each face
- Add midpoint on each edge
- Connect all new vertices

Catmull-Clark vertex update rules (quad mesh)

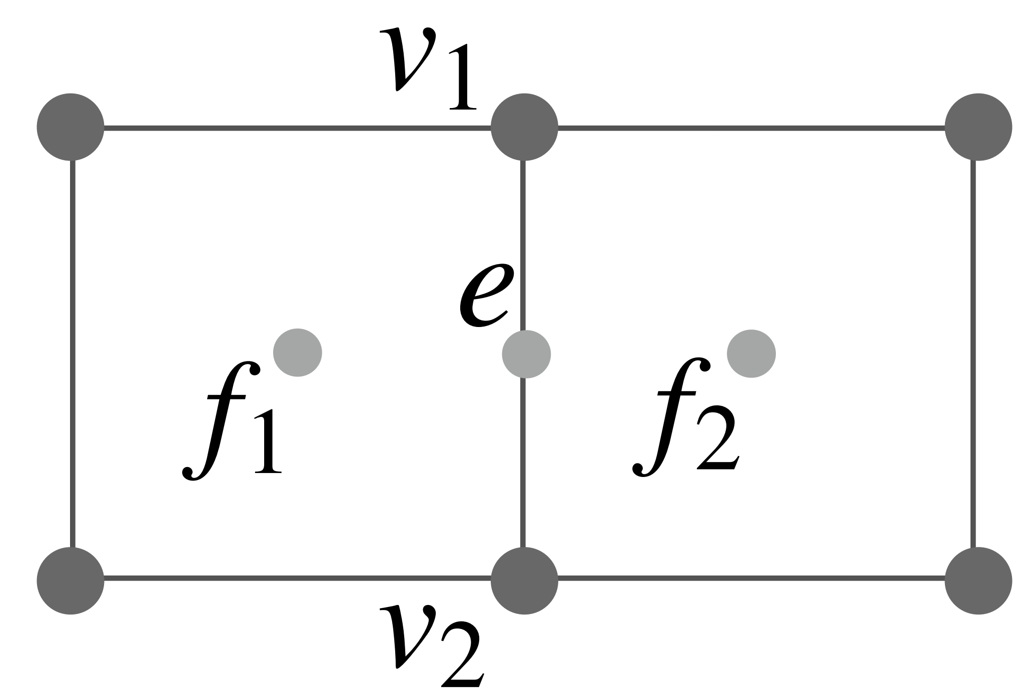
Face point

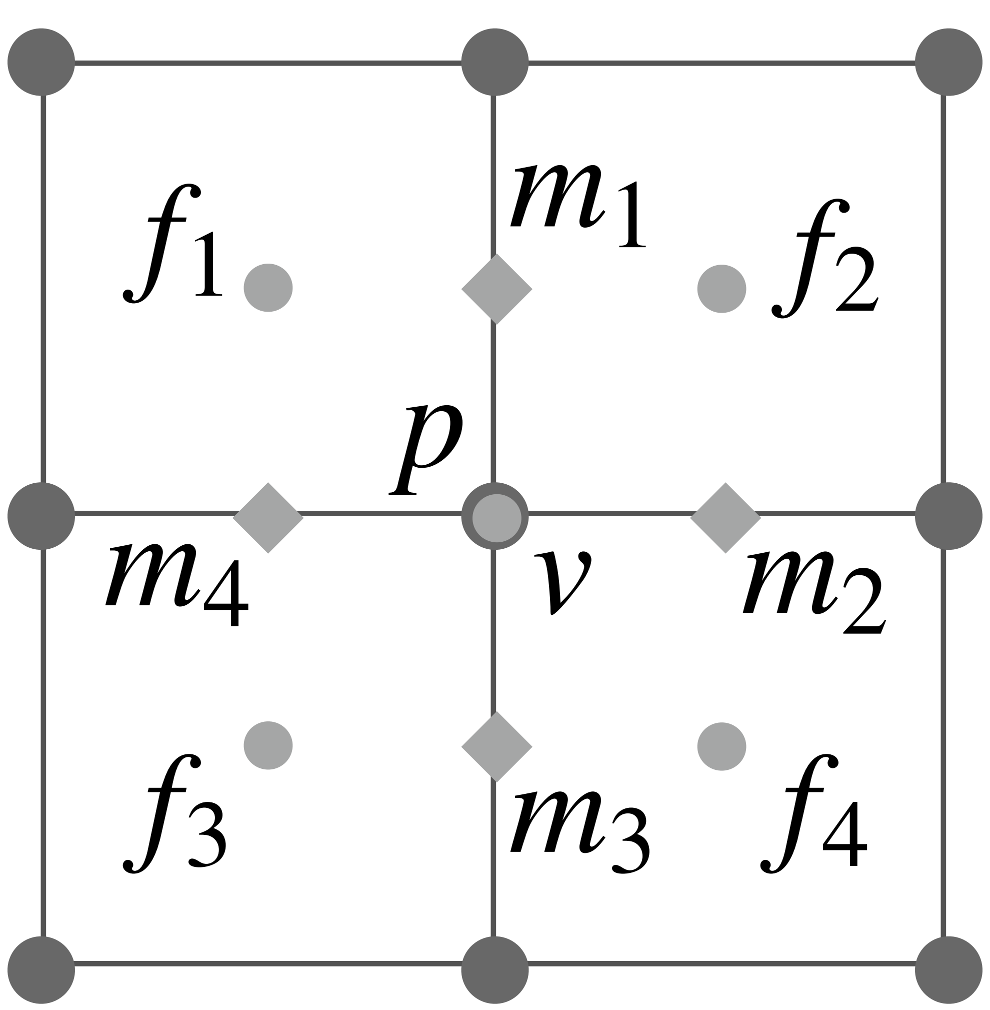


$$f = \frac{v_1 + v_2 + v_3 + v_4}{4}$$

$$e = \frac{v_1 + v_2 + f_1 + f_2}{4}$$

Edge point





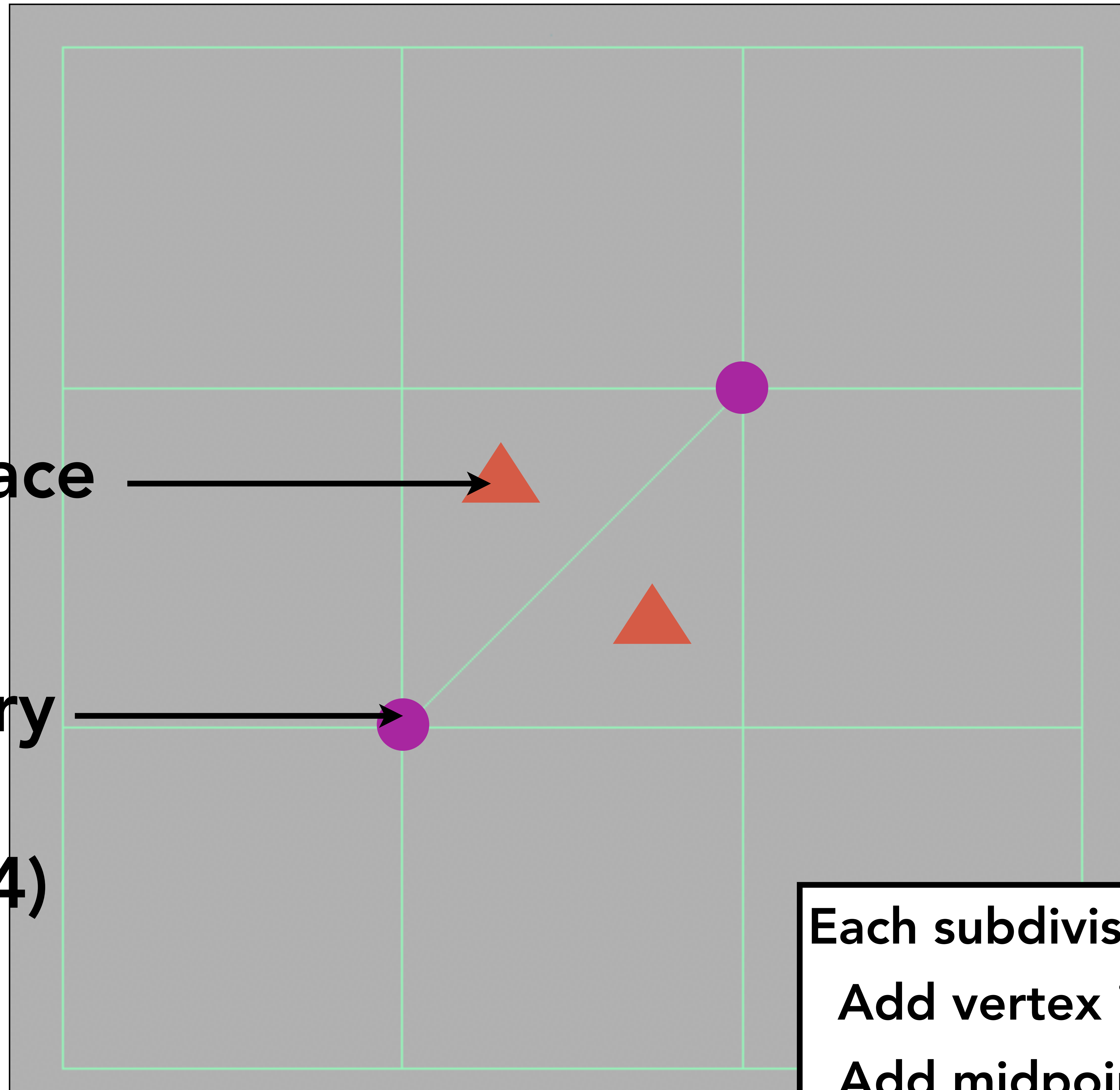
Vertex point

$$v = \frac{f_1 + f_2 + f_3 + f_4 + 2(m_1 + m_2 + m_3 + m_4) + 4p}{16}$$

m midpoint of edge, not "edge point"

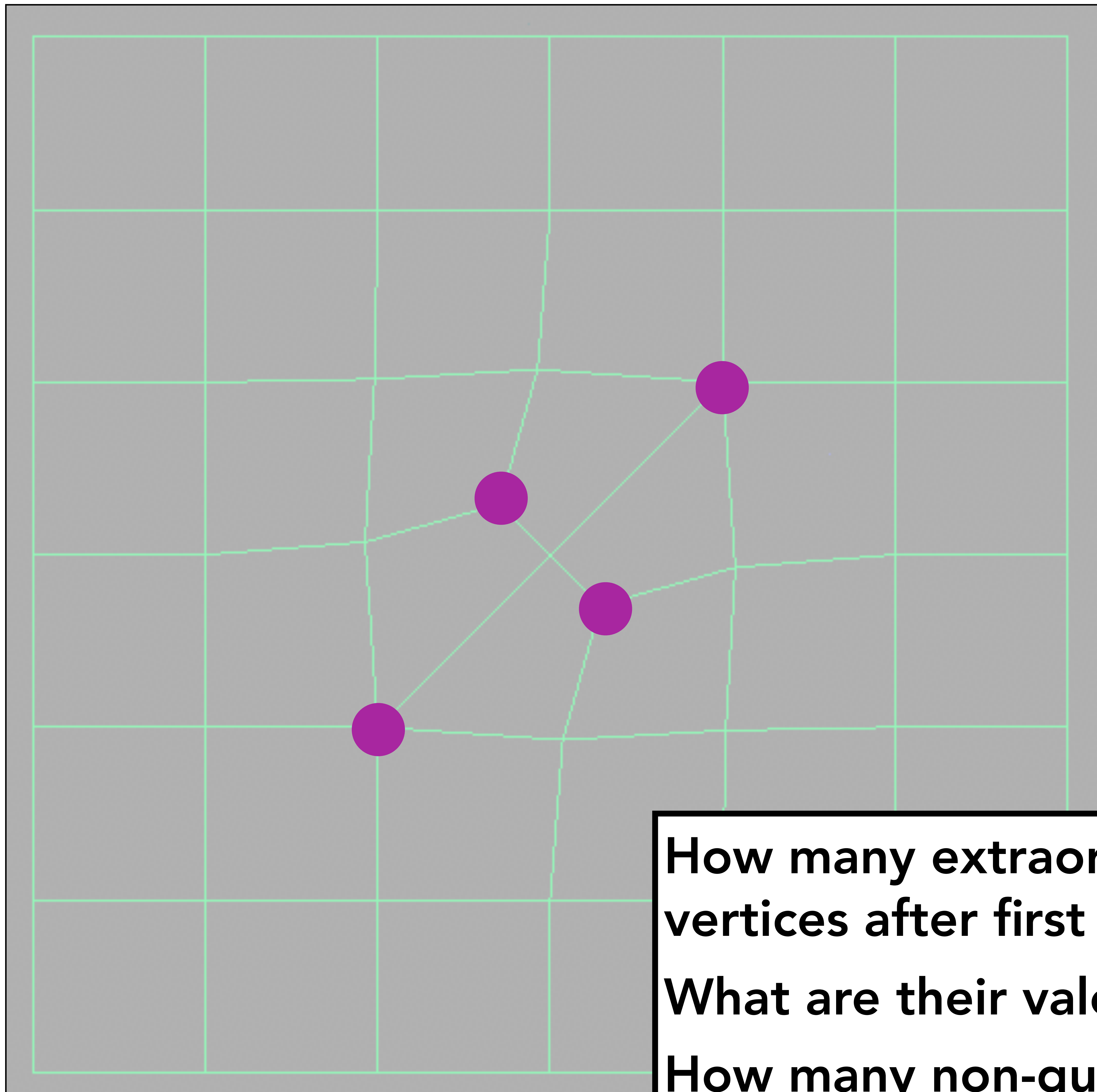
p old "vertex point"

Catmull-Clark subdivision (general mesh)



Each subdivision step:
- Add vertex in each face
- Add midpoint on each edge
- Connect all new vertices

Catmull-Clark subdivision (general mesh)

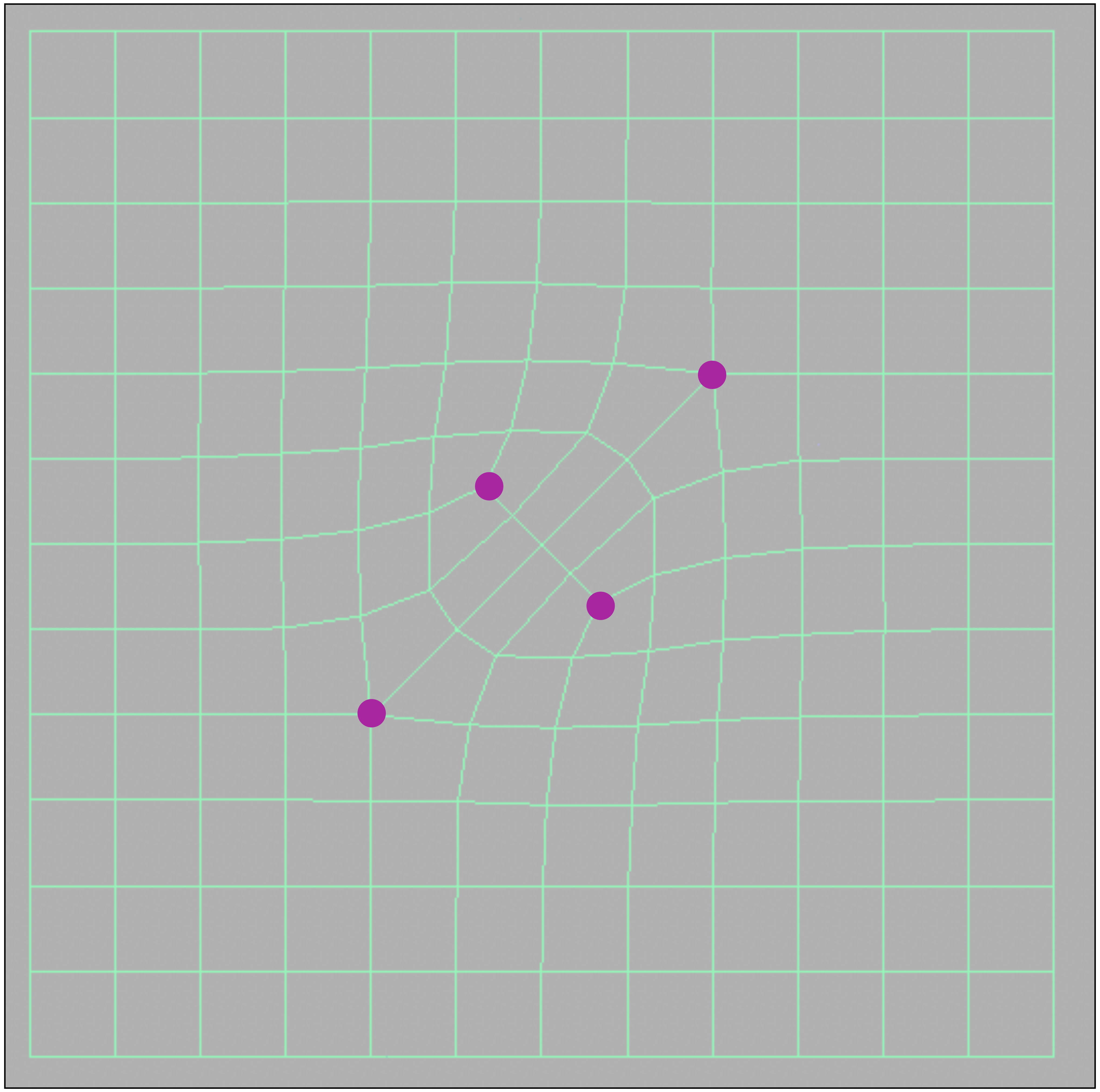


How many extraordinary vertices after first subdivision?

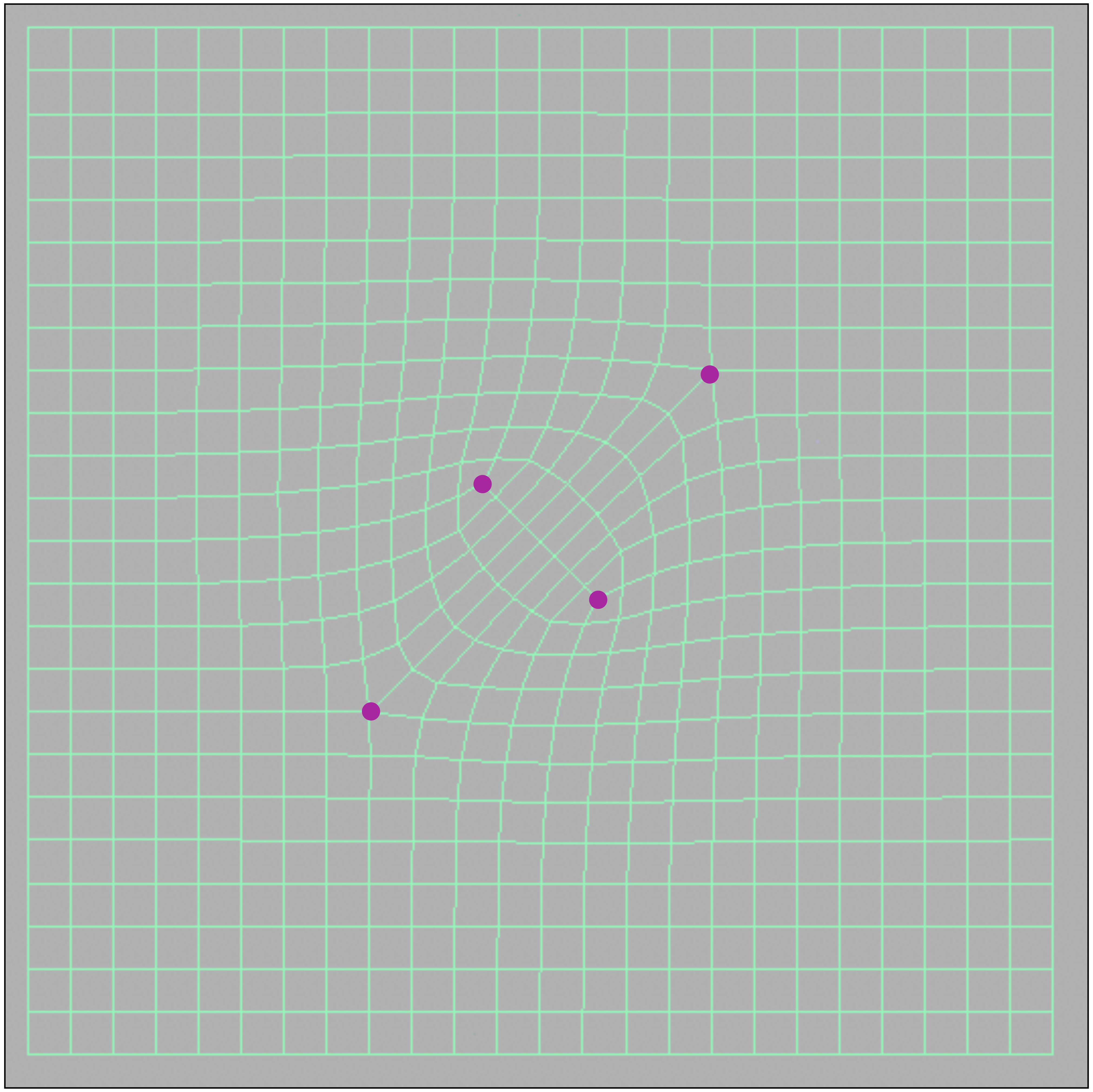
What are their valences?

How many non-quad faces?

Catmull-Clark subdivision (general mesh)



Catmull-Clark subdivision (general mesh)



Catmull-Clark vertex update rules (general mesh)

$$f = \text{average of surrounding vertices}$$

$$e = \frac{f_1 + f_2 + v_1 + v_2}{4}$$

These rules reduce to earlier quad rules for ordinary vertices / faces

$$v = \frac{\bar{f}}{n} + \frac{2\bar{m}}{n} + \frac{p(n-3)}{n}$$

$$\bar{m} = \text{average of adjacent midpoints}$$
$$\bar{f} = \text{average of adjacent face points}$$
$$n = \text{valence of vertex}$$
$$p = \text{old ''vertex'' point}$$

Continuity of Catmull-Clark surface

- At extraordinary points
  - Surface is at least C^1 continuous

- Everywhere else ("ordinary" regions)
  - Surface is C^2 continuous

What about sharp creases?

From Pixar Short, "Geri's Game"

Hand is modeled as a Catmull Clark surface with creases between skin and fingernail

What about sharp creases?

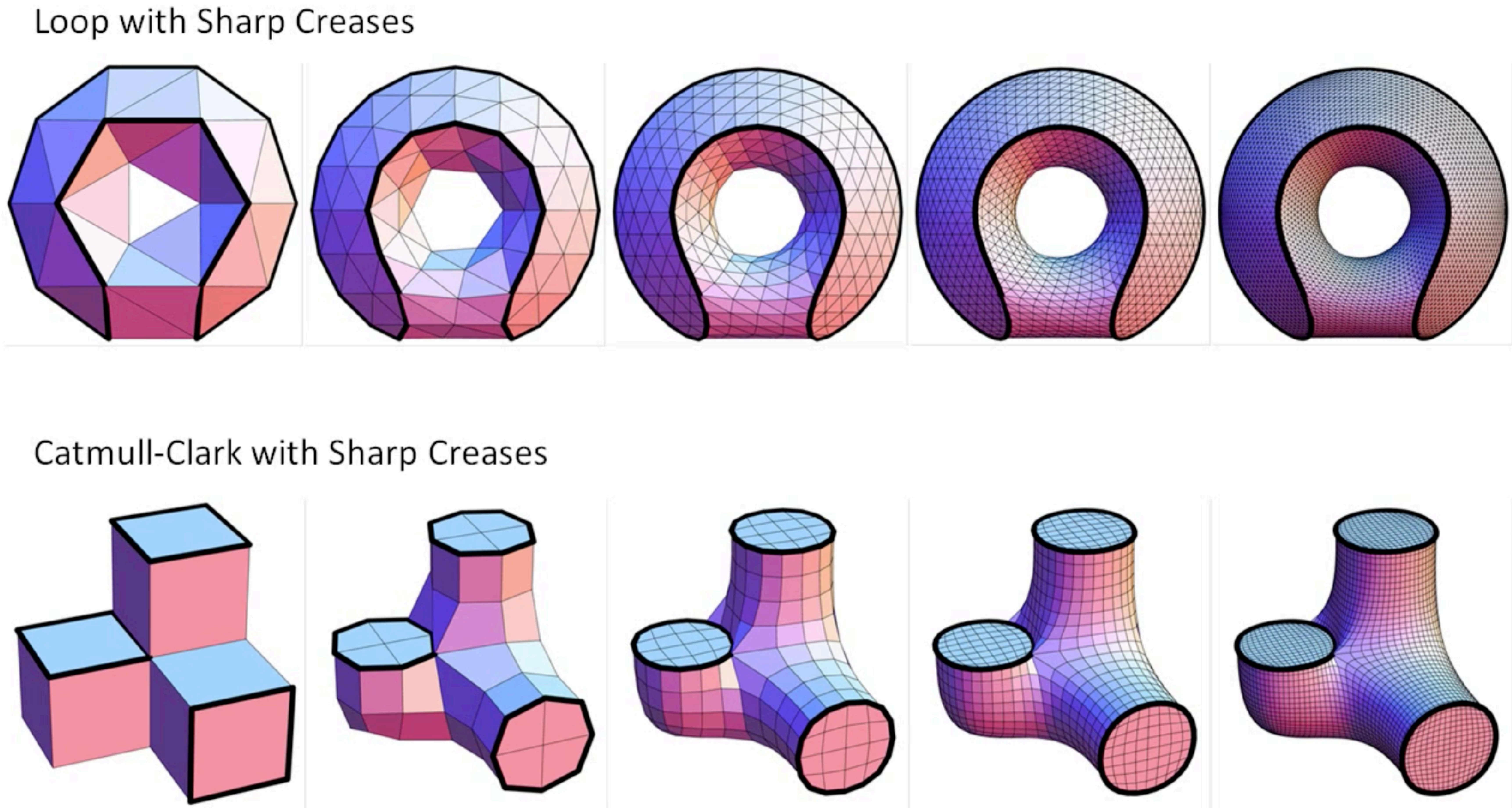
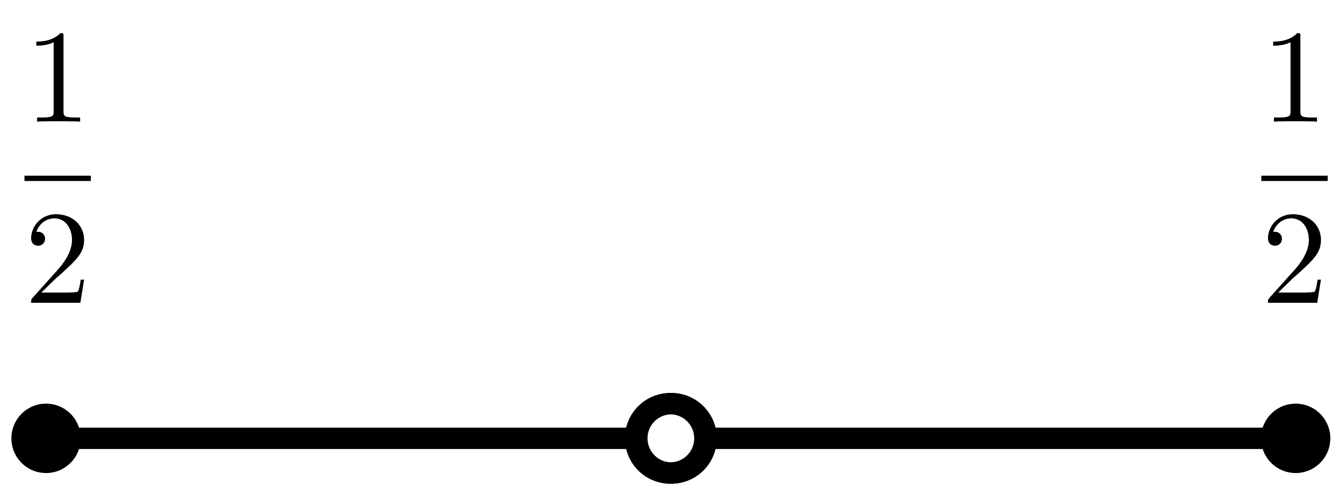


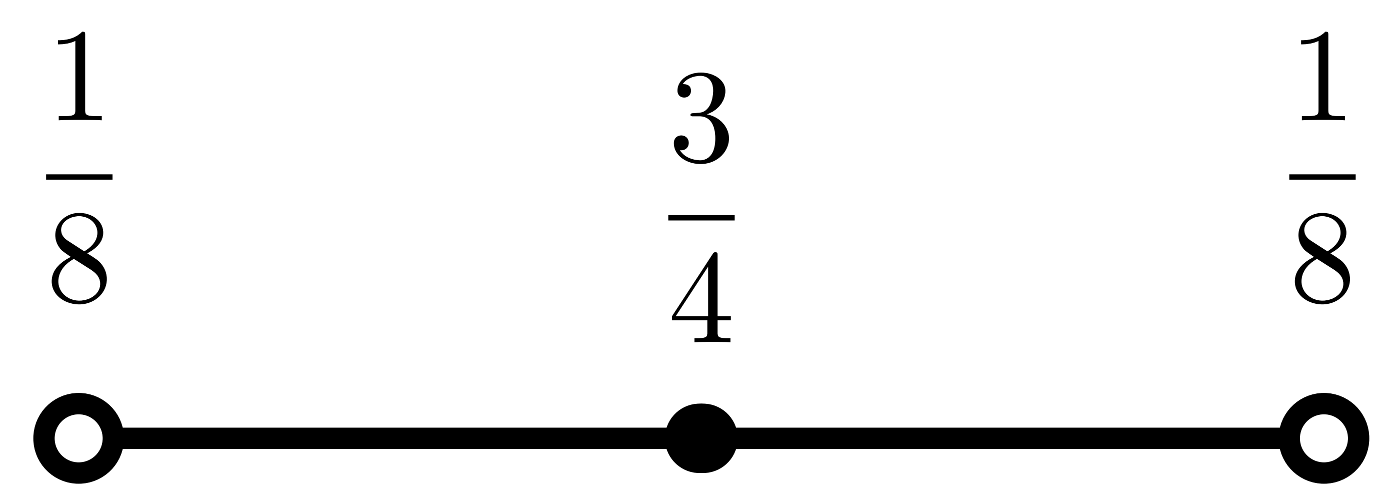
Figure from: Hakenberg et al. Volume Enclosed by Subdivision Surfaces with Sharp Creases

Creases and boundaries

- **Can create creases in subdivision surfaces by marking certain edges as "sharp". Surface boundary edges can be handled the same way**
  - **Use different subdivision rules for vertices along these "sharp" edges**



Insert new midpoint vertex, weights as shown



Update existing vertices, weights as shown

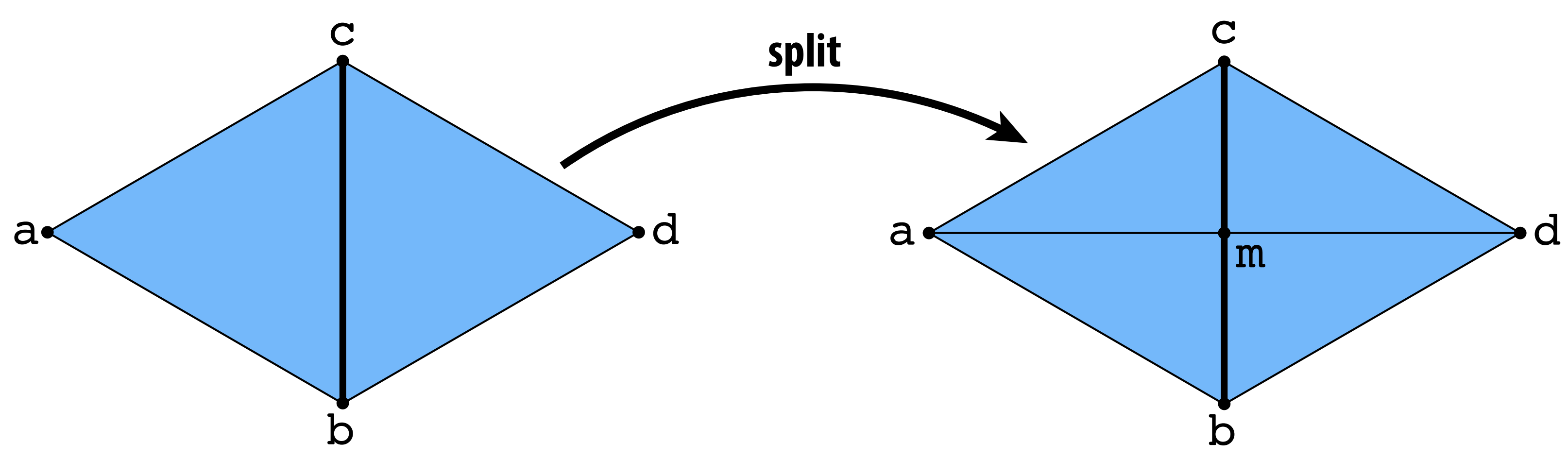
Subdivision in action ("Geri's Game", Pixar)

- Subdivision used for entire character:
  - Hands and head
  - Clothing, tie, shoes

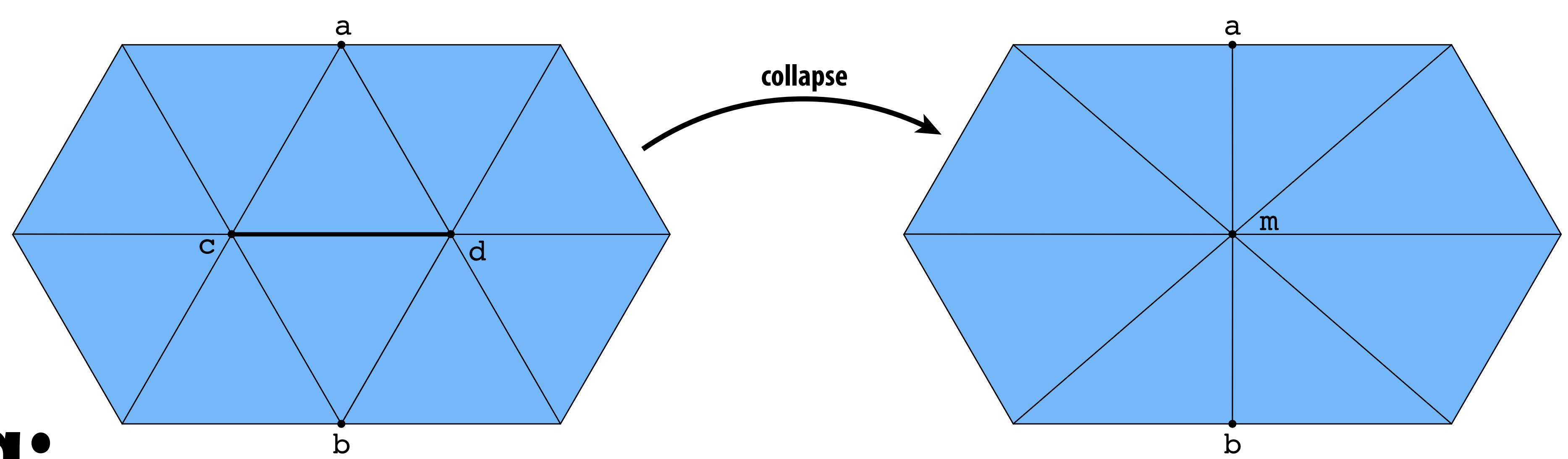
Mesh simplification — downsampling

How do we resample meshes? (reminder)

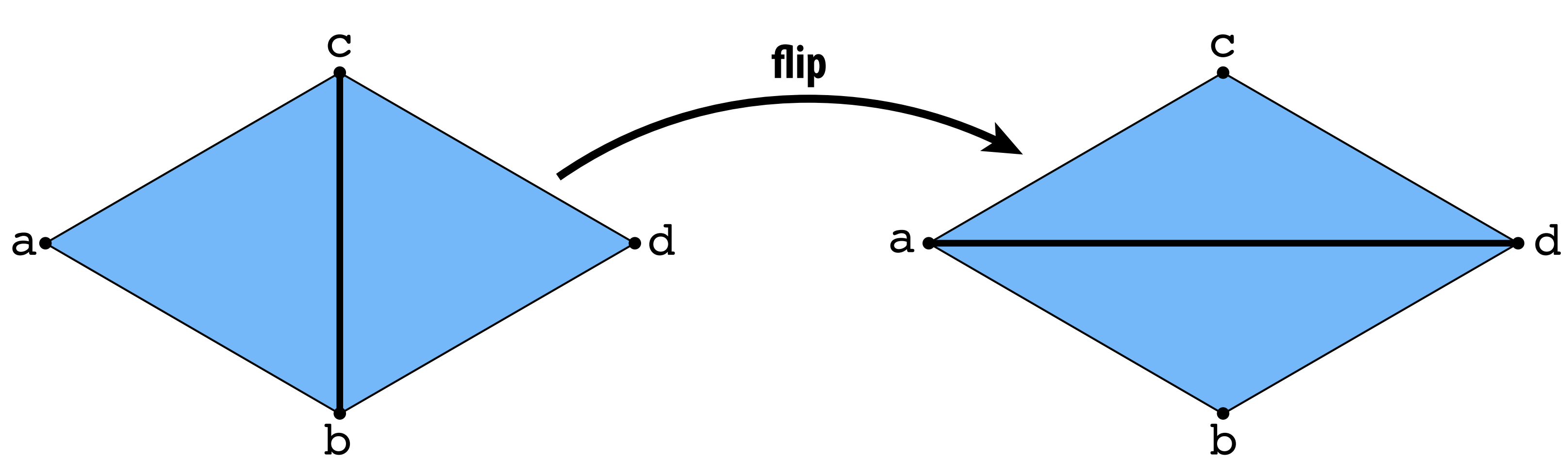
- **Edge split is (local) upsampling:**



- **Edge collapse is (local) downsampling:**



- **Edge flip is (local) resampling:**



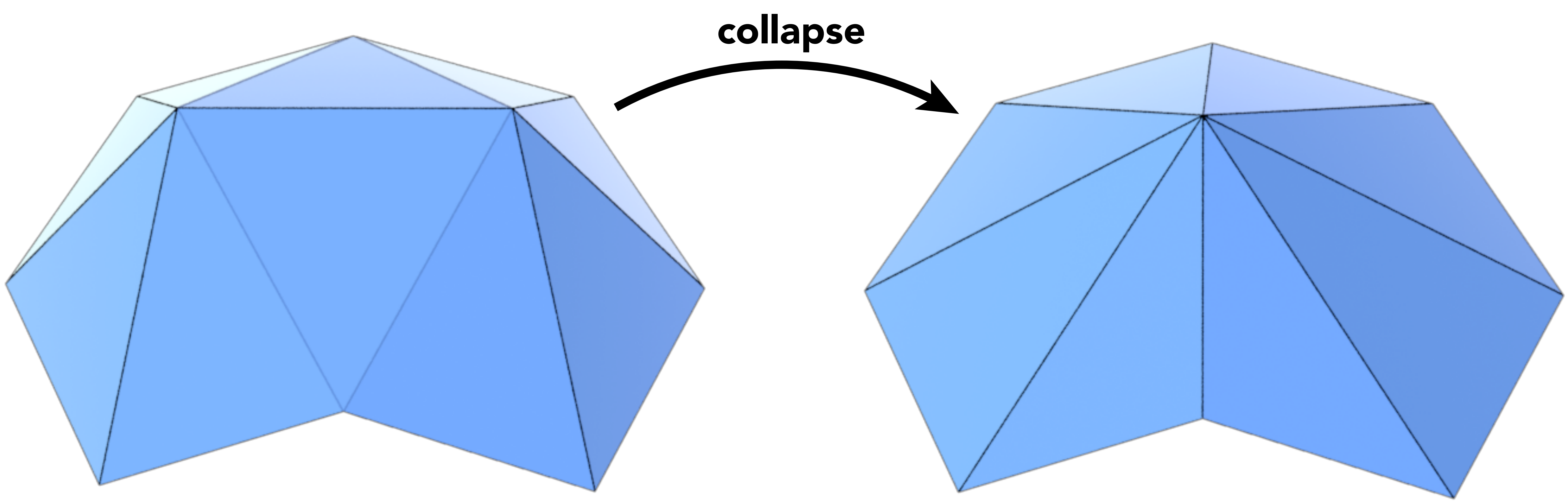
- **Still need to intelligently decide which edges to modify!**

Mesh simplification

- Goal: reduce number of mesh elements while maintaining overall shape

Estimate: error introduced by collapsing an edge?

How much geometric error is introduced by collapsing an edge?



Sketch of Quadric Error Mesh Simplification

Simplification via quadric error

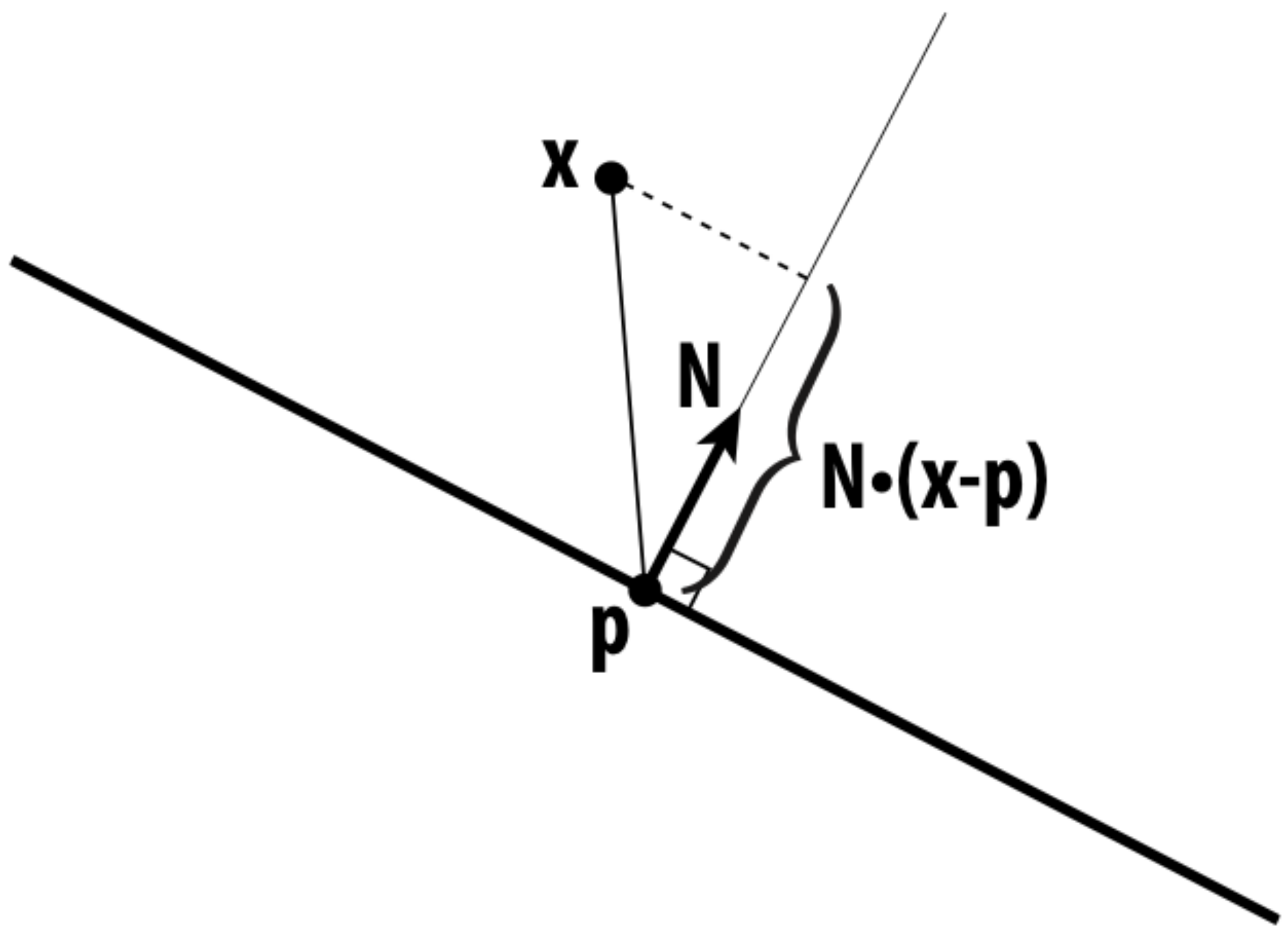
- Iteratively collapse edges
- Which edges? Assign score with quadric error metric*
  - Approximate distance to surface as sum of squared distances to planes containing nearby triangles
  - Iteratively collapse edge with smallest score
  - Greedy algorithm... great results!

* (Garland & Heckbert 1997)

Review: point-to-plane distance

Signed distance to plane with normal N passing through point p?

=> N • (x – p)

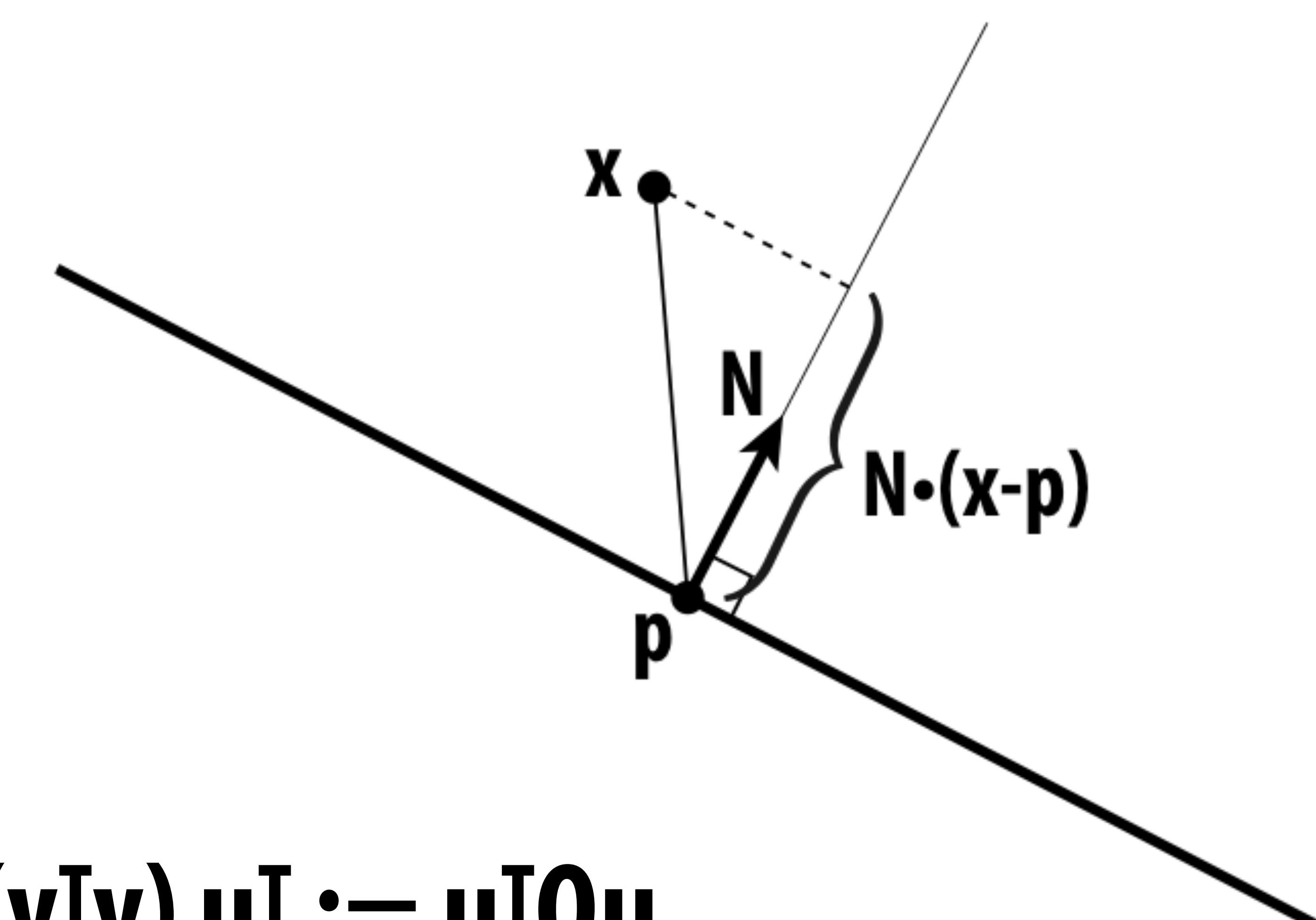


Quadric error matrix (encodes squared distance)

- Suppose we have:
  - a query point (x,y,z)
  - a normal (a,b,c)
  - an offset $d := -(x_p, y_p, z_p) \cdot (a,b,c)$

$$Q = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$$

- Then in homogeneous coordinates, let
  - $u := (x,y,z,1)$
  - $v := (a,b,c,d)$



- Signed distance to plane is then
  $D = uv^T = vu^T = ax+by+cz+d$

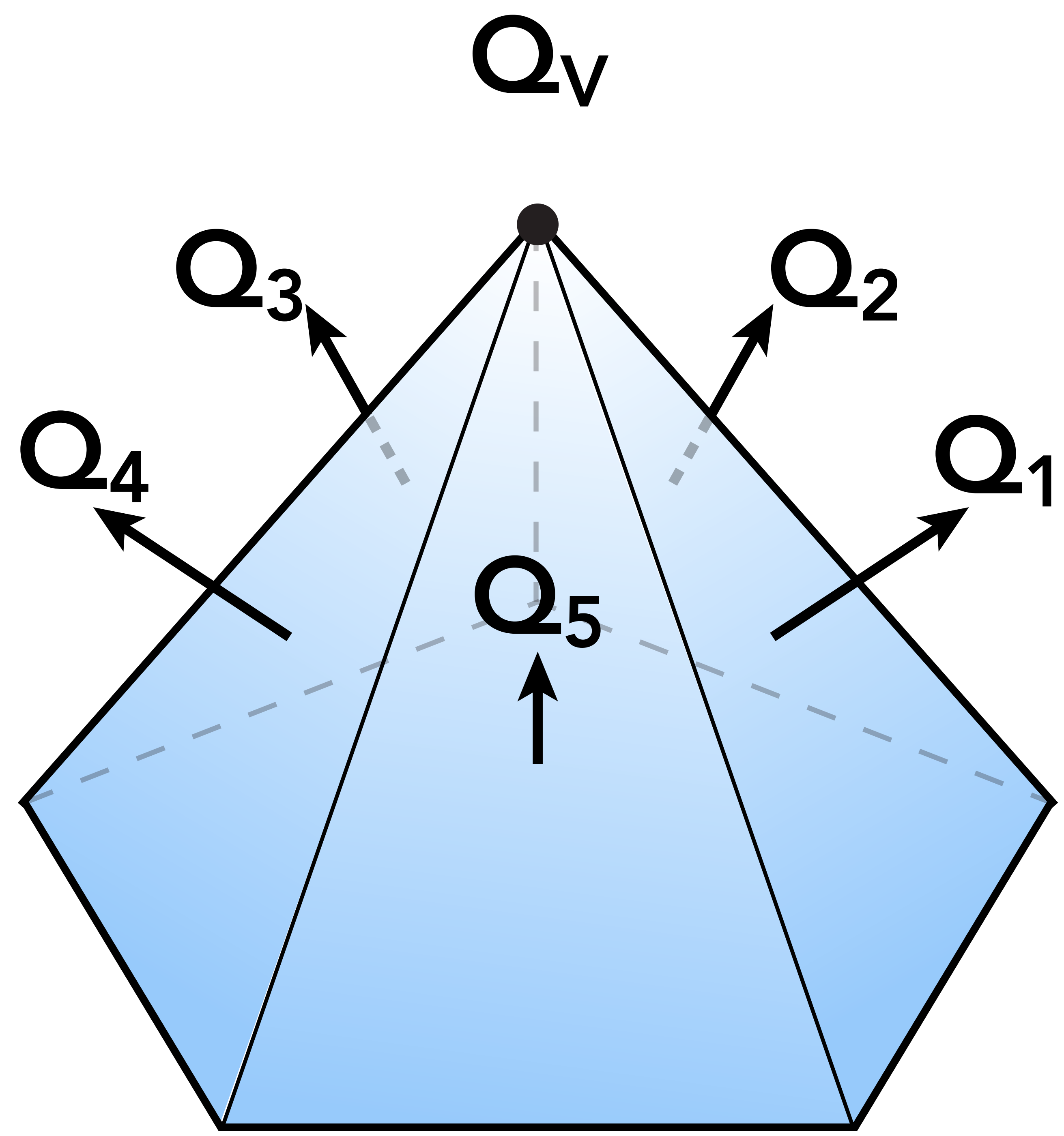
- Squared distance is $D^2 = (uv^T)(vu^T) = u\,(v^Tv)\,u^T := u^TQu$

- Distance is 2nd degree ("quadric") polynomial in x,y,z

Quadric error at mesh vertex

Heuristic: error at vertex V is sum of squared distances to triangles connected to V

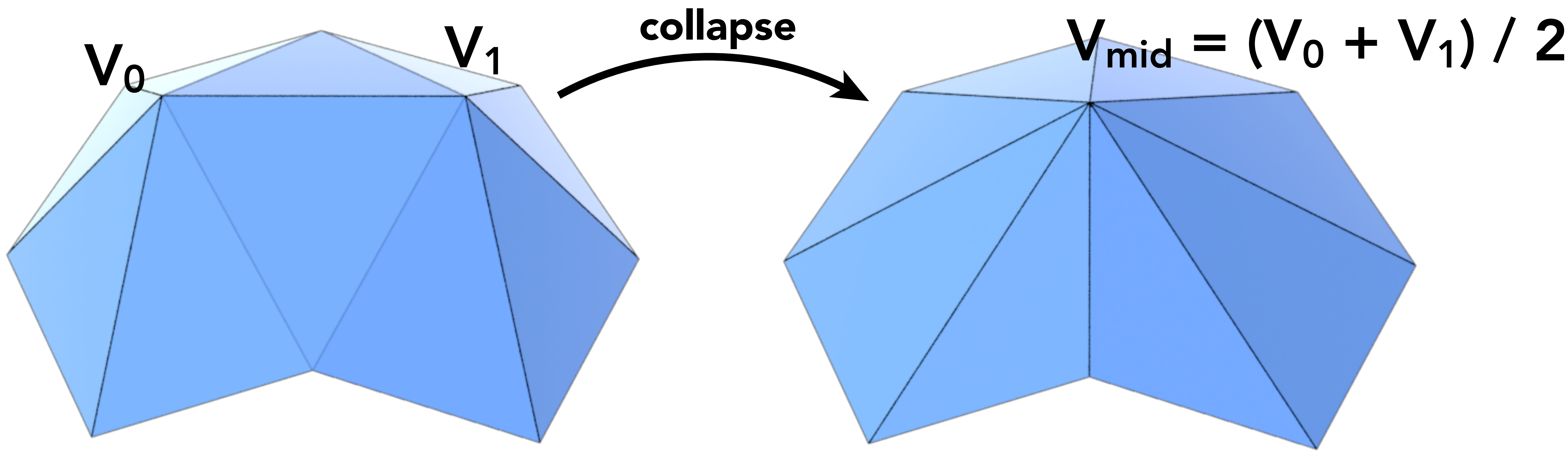
Encode this as a single quadric matrix per vertex that is the sum of quadric error matrices for all triangles



$$Q_V = \sum_{i=1}^{N} Q_i$$

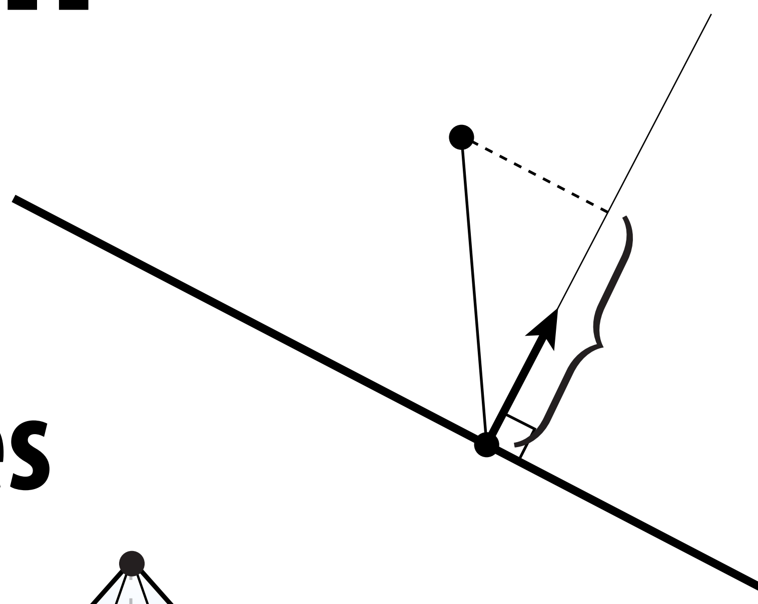
Cost of edge collapse

- How much does it cost to collapse an edge?
- Idea: compute edge midpoint V_{mid}, measure quadric error at this point
- Error at V_{mid} given by $v_{mid}^T(Q_0 + Q_1)v_{mid}$
- Intuition: cost is sum of squared differences to original position of triangles now touching V_{mid}

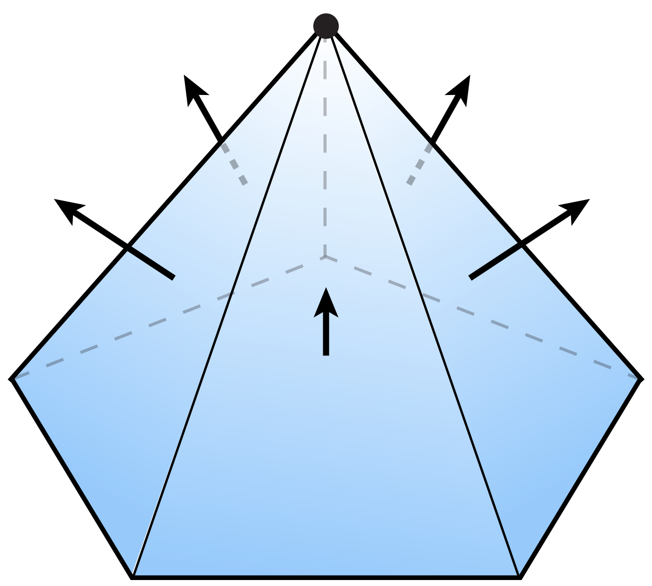


- Better idea: choose point on edge (not necessarily the midpoint) that minimizes quadric error
- More details: Garland & Heckbert 1997

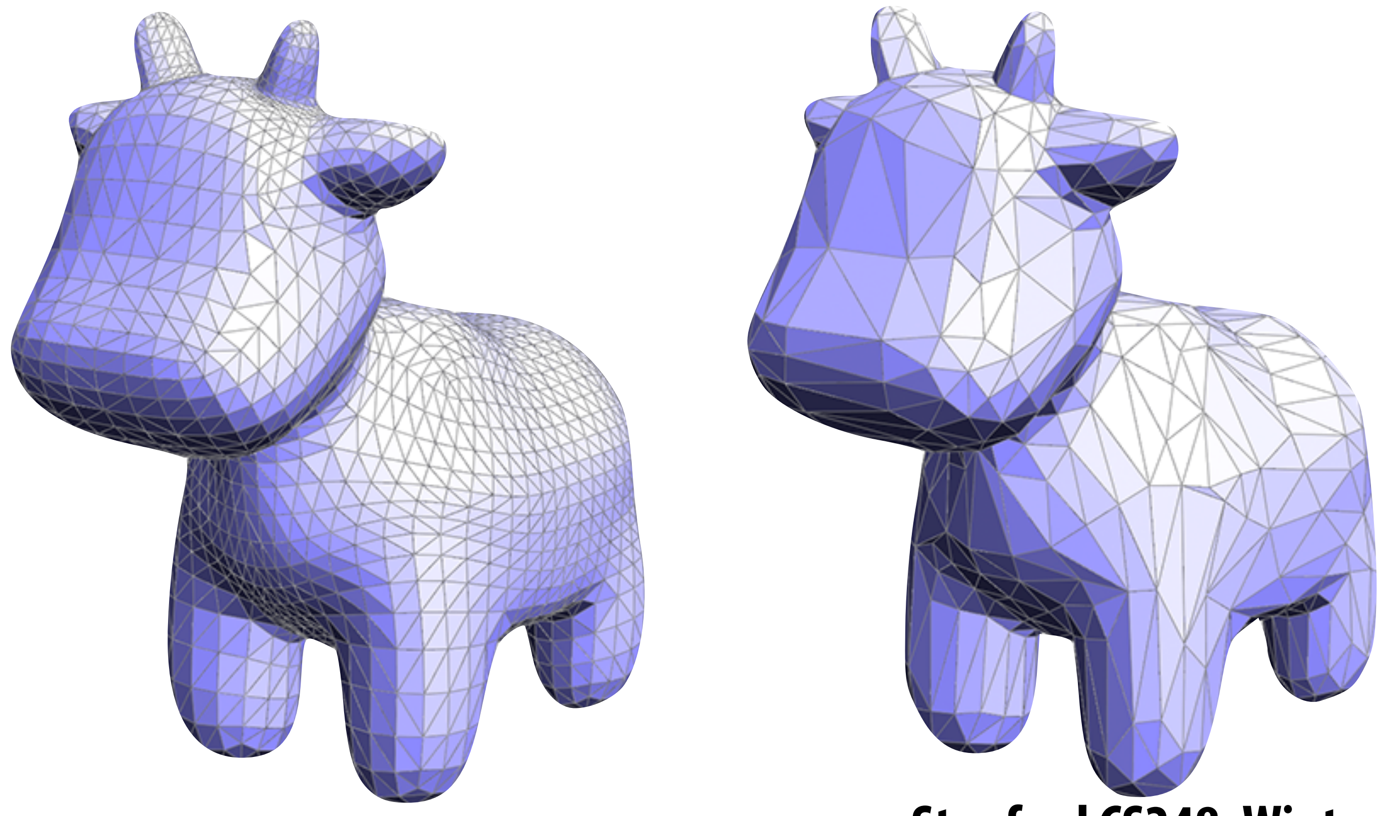
Quadric error simplification: algorithm



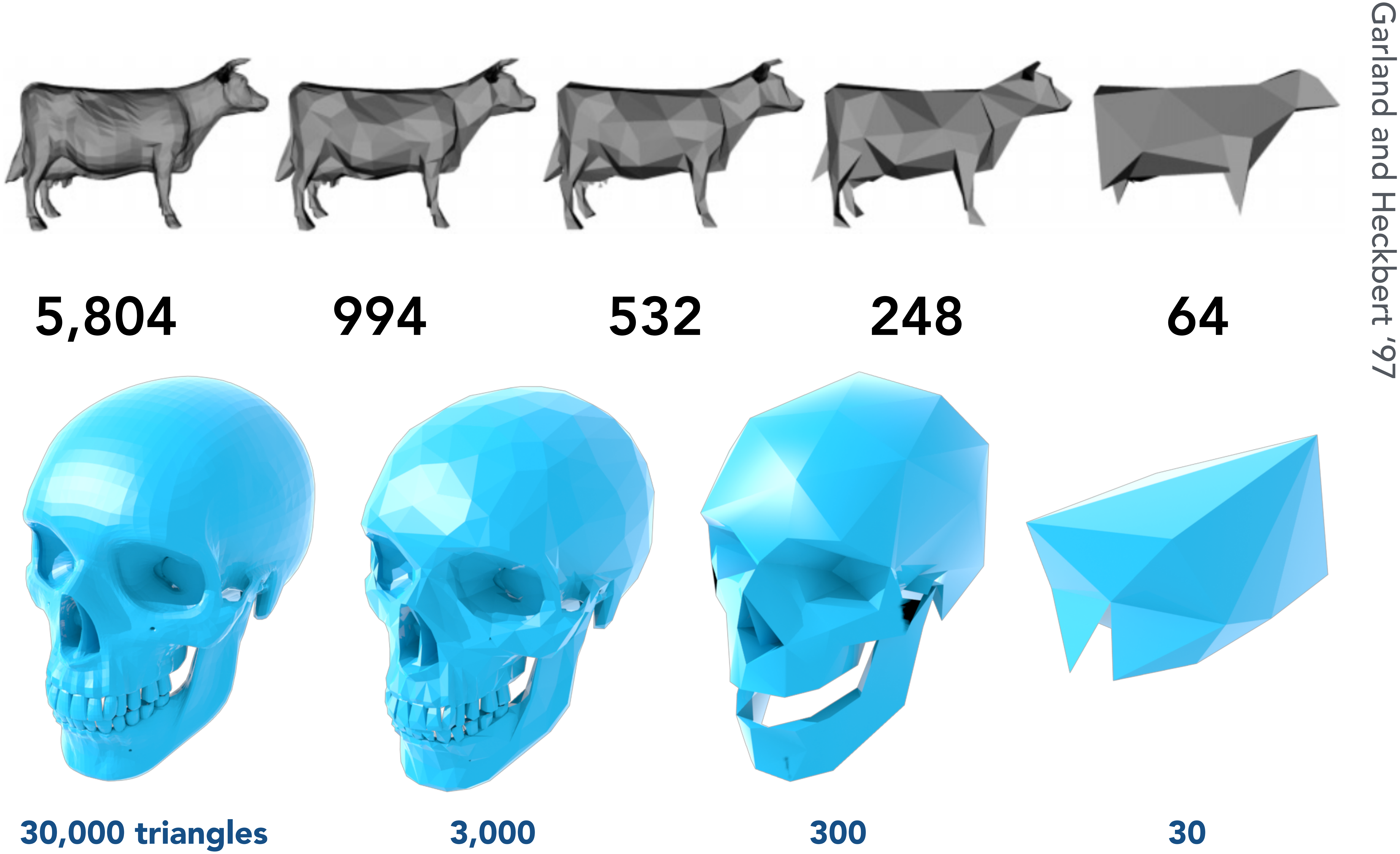
- Compute quadric error matrix Q for each triangle's plane
- Set Q at each vertex to sum of Q's from neighbor triangles
- Set Q at each edge to sum of Q's at endpoints



- Find point at each edge minimizing quadric error
- Until we reach target # of triangles:
  - collapse edge (i,j) with smallest cost to get new vertex m
  - add Q_i and Q_j to get quadric Q_m at vertex m
  - update cost of edges touching vertex m

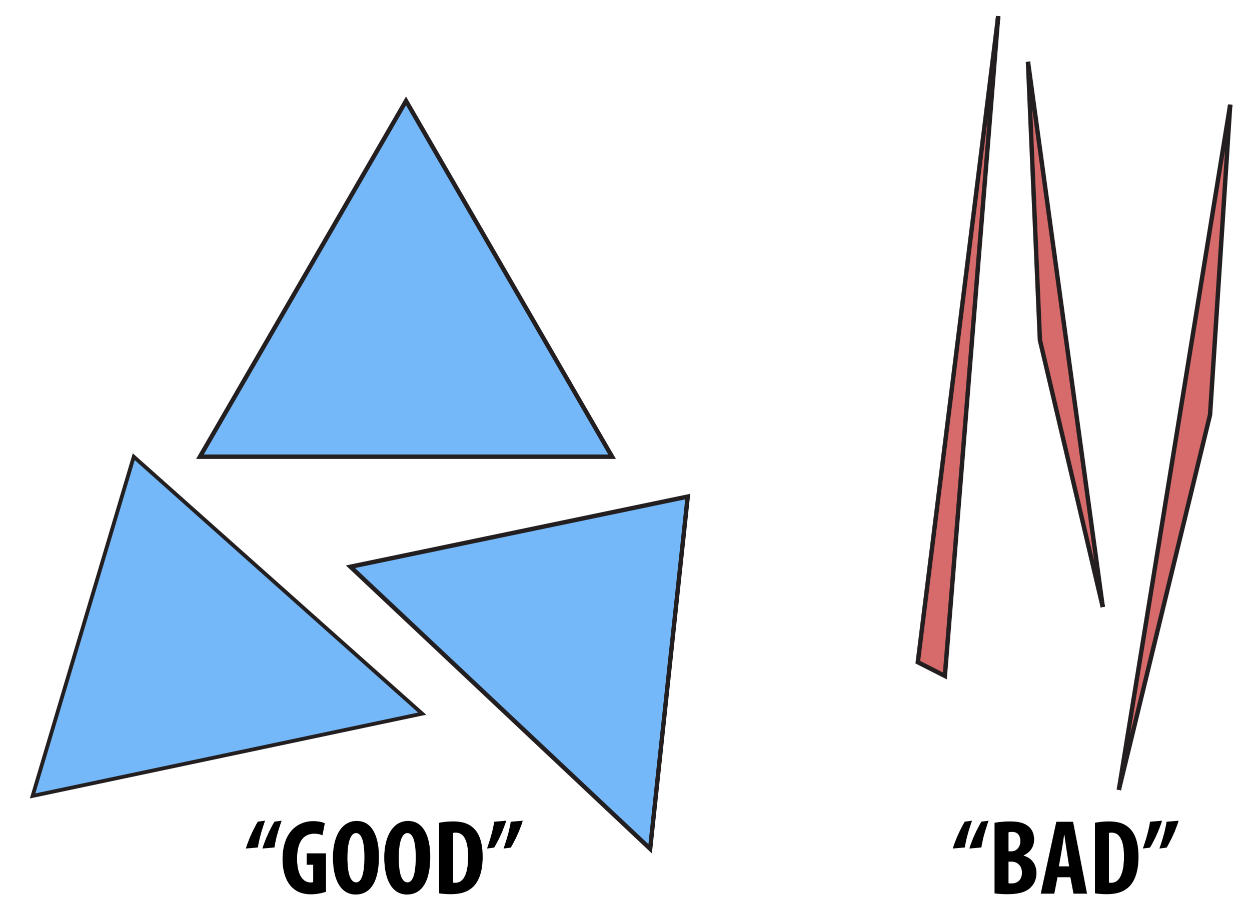


Quadric error mesh simplification

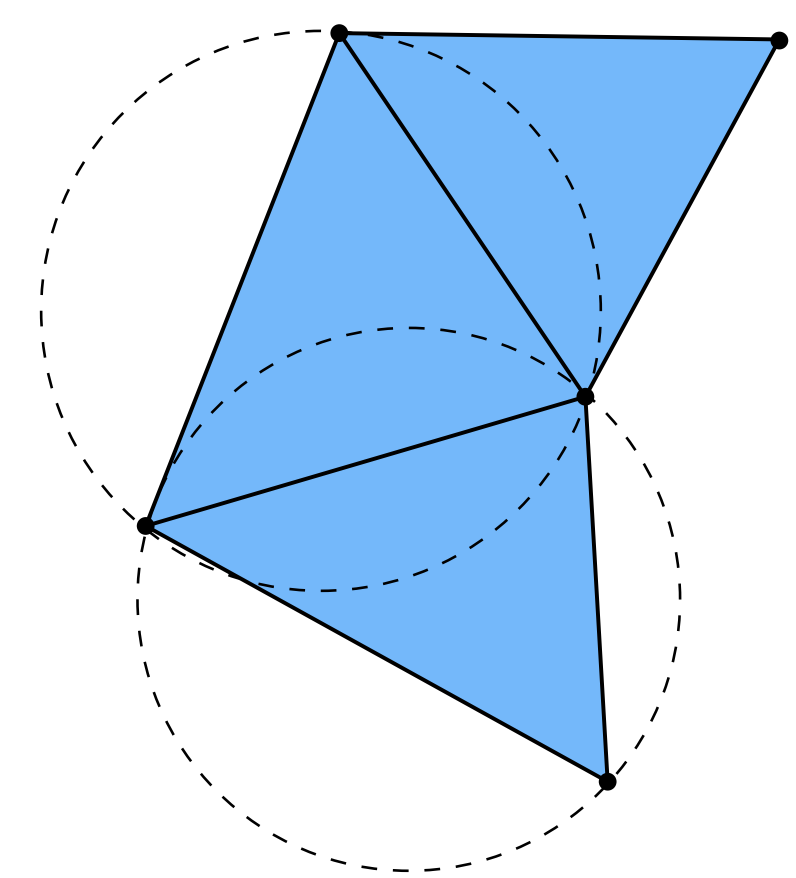


Mesh Regularization

What makes a “good” triangle mesh?



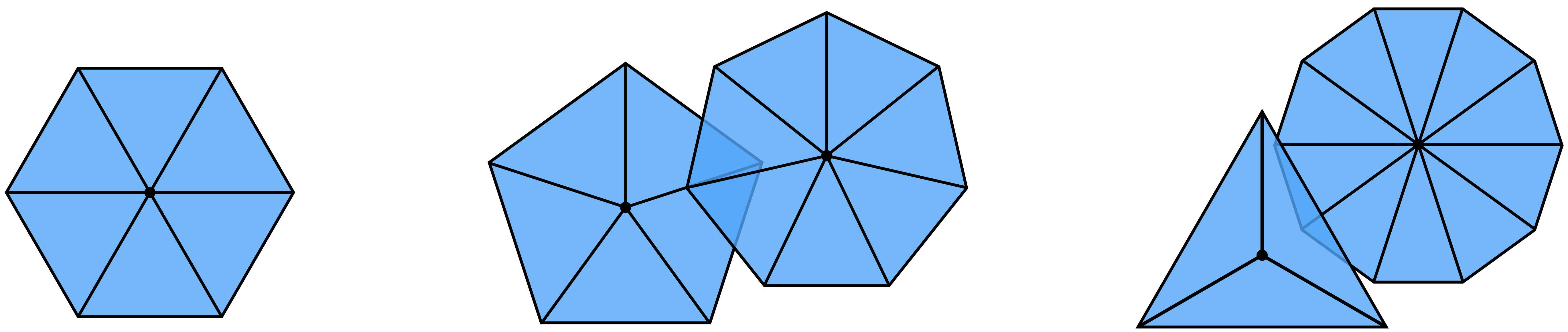
- One rule of thumb: triangle shape
- More specific condition: Delaunay
  - “Circumcircle interiors contain no vertices.”
- Not always a good condition, but often*
  - Good for simulation
  - Not always best for shape approximation



*See Shewchuk, “What is a Good Linear Element”

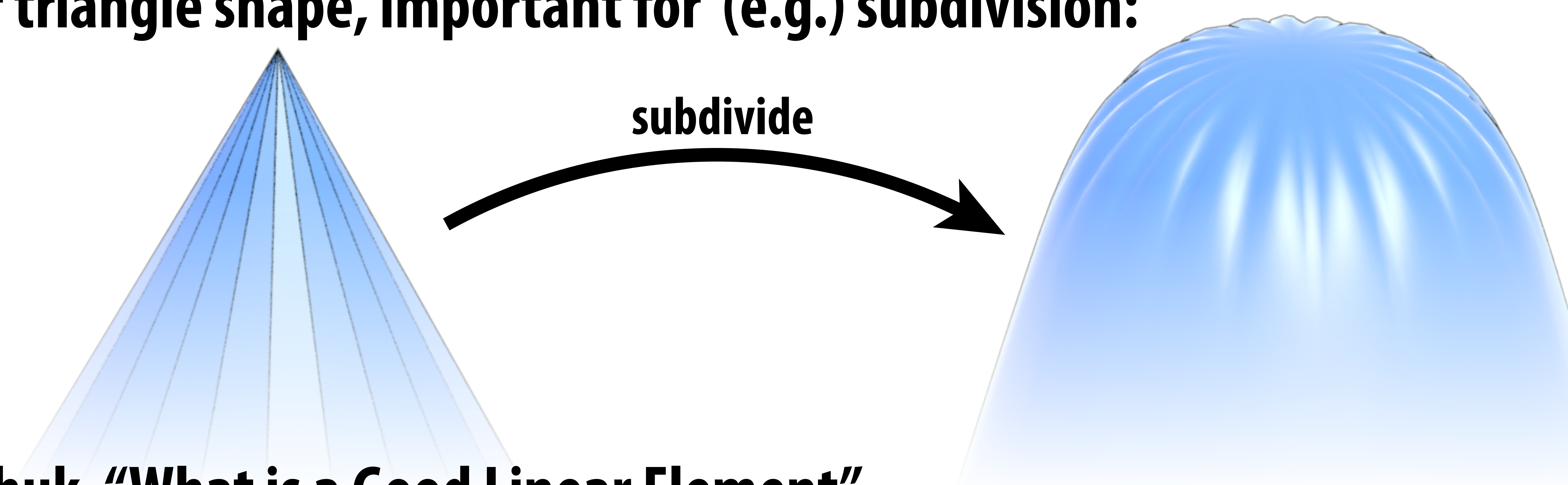
What else constitutes a good mesh?

- **Rule of thumb: regular vertex degree**
- **Triangle meshes: ideal is every vertex with valence 6:**



"GOOD" "OK" "BAD"

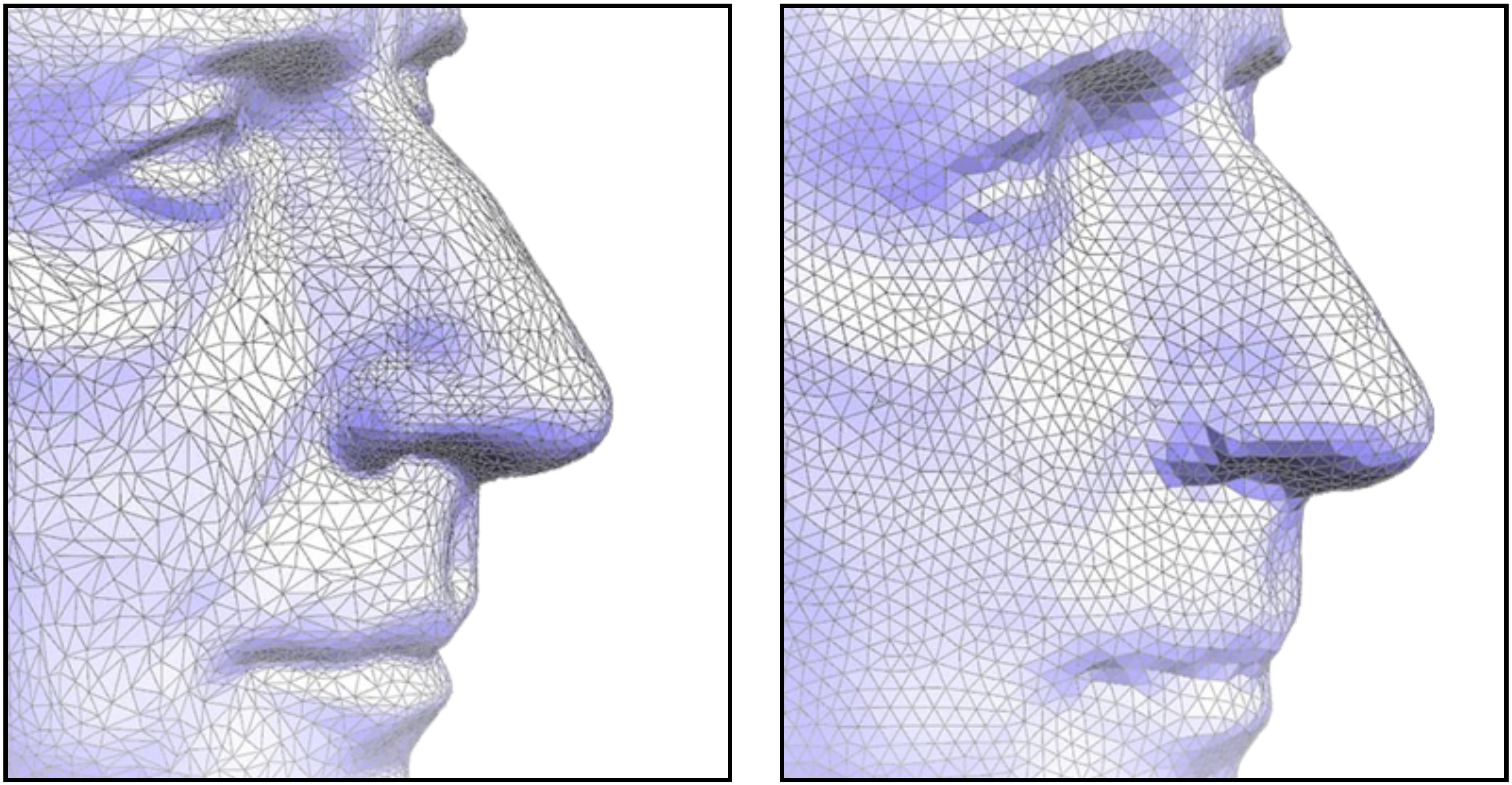
Why? Better triangle shape, important for (e.g.) subdivision:



***See Shewchuk, "What is a Good Linear Element"**

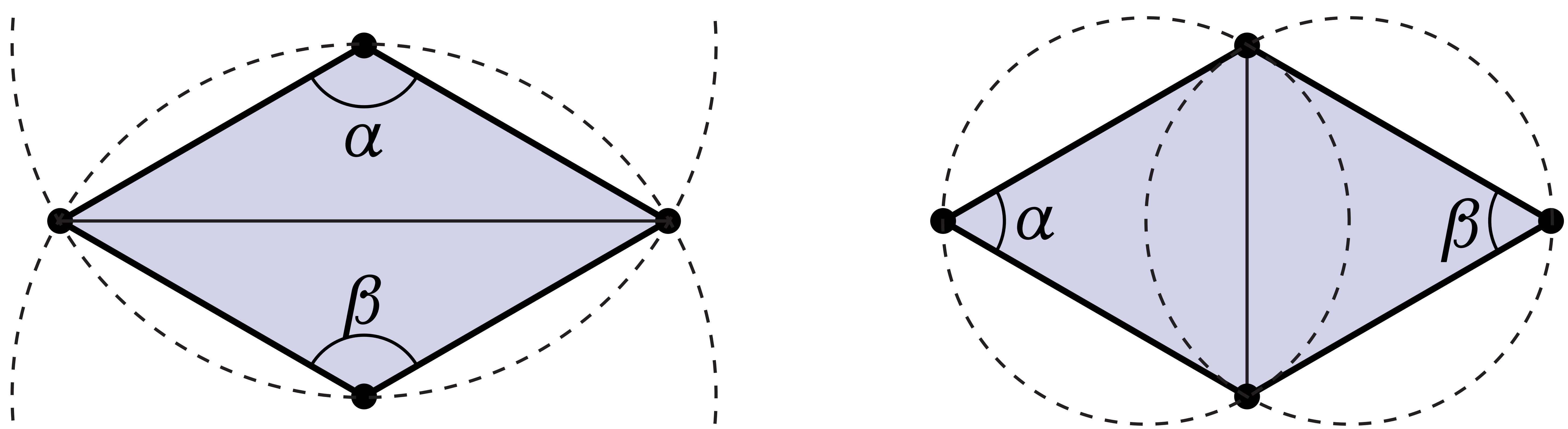
Isotropic remeshing

Goal: try to make triangles uniform in shape and size



How do we make a mesh "more delaunay"?

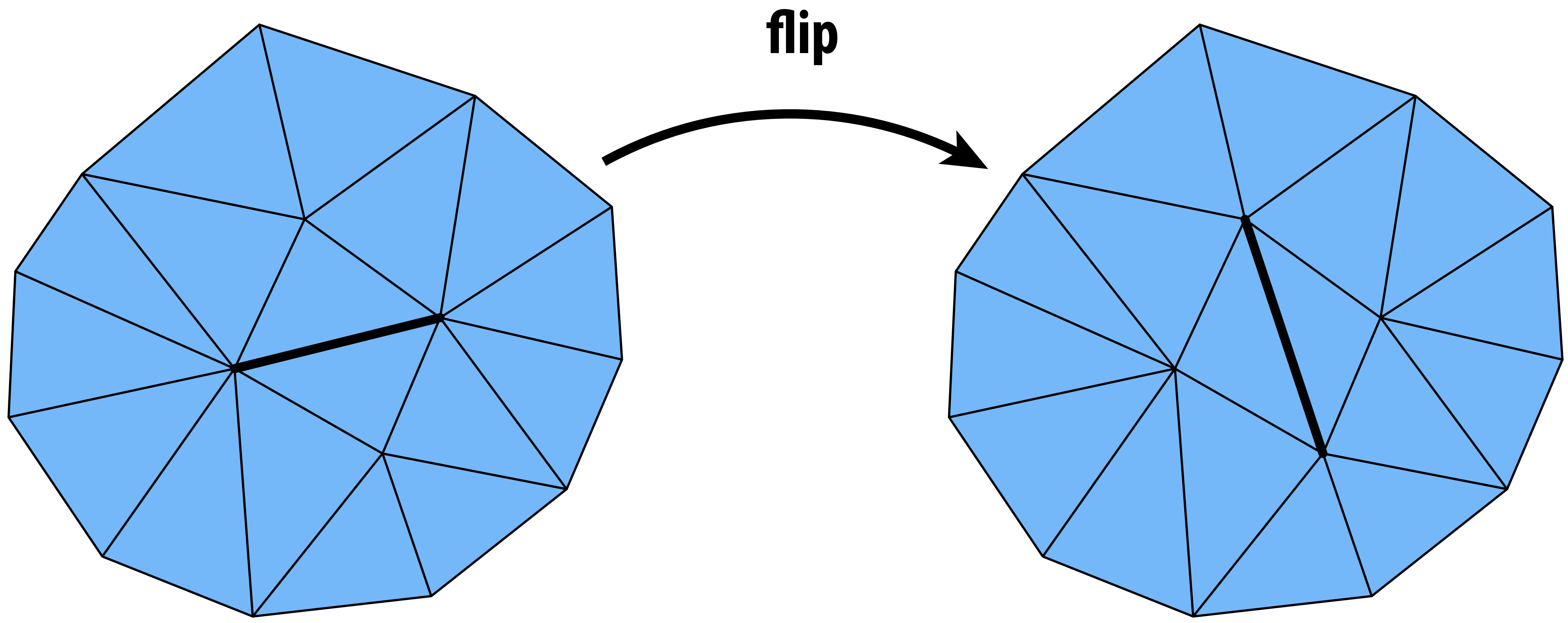
- **Already have a good tool: edge flips!**
- **If $\alpha+\beta > \pi$, flip it!**



- **In practice: a simple, effective way to improve mesh quality**

How do we improve degree?

- Edge flips!
- If total deviation from degree 6 gets smaller, flip it!

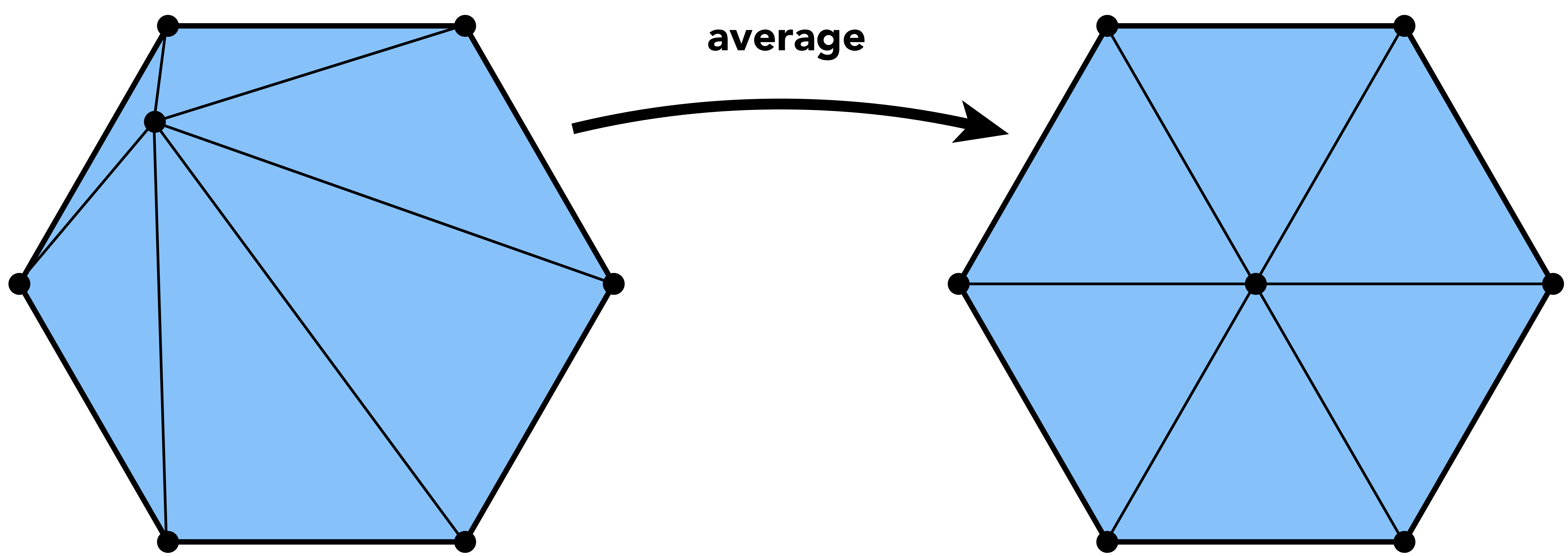


Iterative edge flipping acts like "discrete diffusion" of degree

No (known) guarantees; works well in practice

How do we make triangles “more round”?

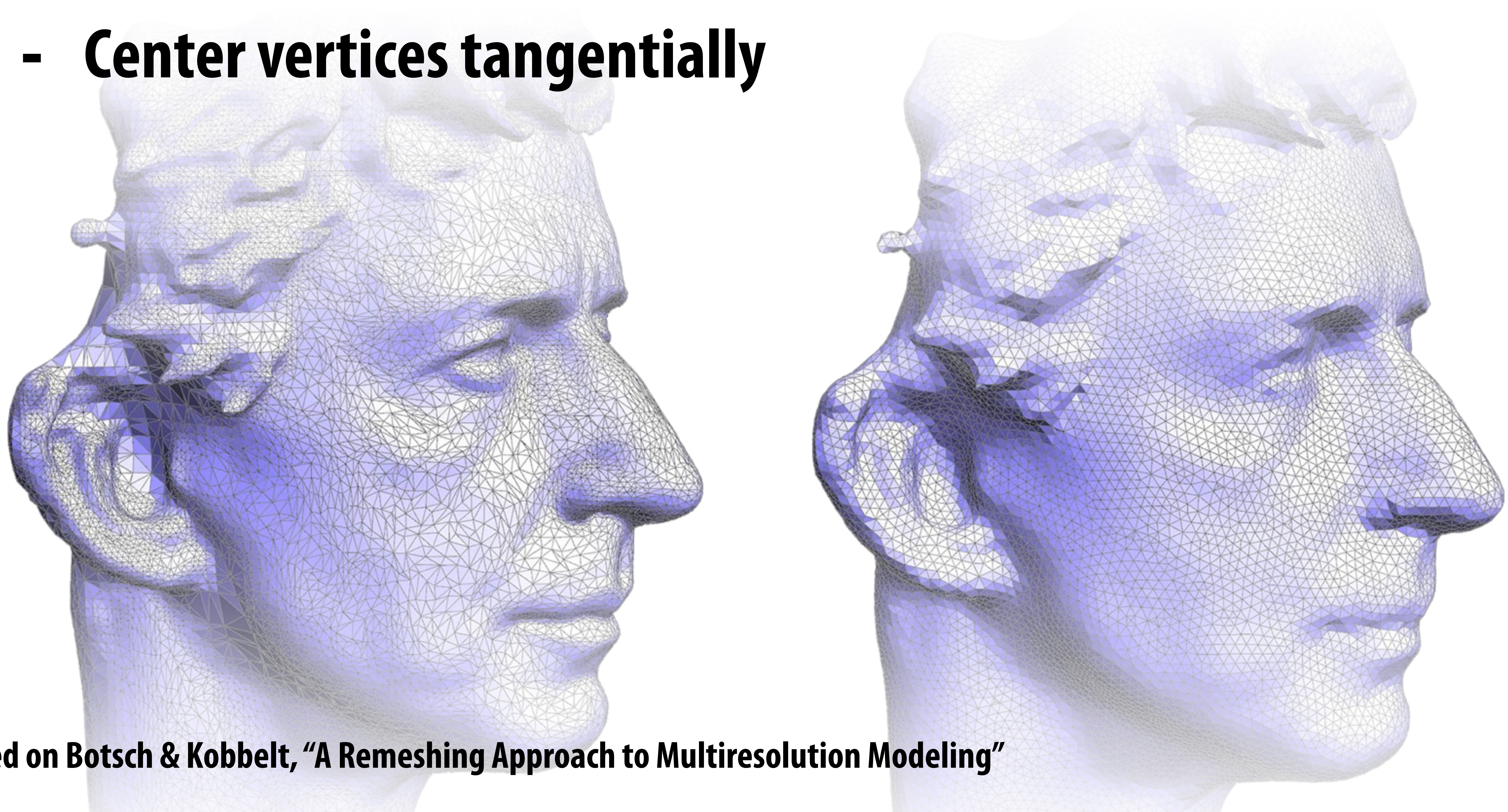
- Delaunay doesn’t mean equilateral triangles
- Can often improve shape by centering vertices:



[See Crane, “Digital Geometry Processing with Discrete Exterior Calculus”]

Isotropic remeshing algorithm*

- **Repeat four steps:**
  - Split edges over 4/3rds mean edge length
  - Collapse edges less than 4/5ths mean edge length
  - Flip edges to improve vertex degree
  - Center vertices tangentially



* Based on Botsch & Kobbelt, "A Remeshing Approach to Multiresolution Modeling"

Things to remember

- Triangle mesh representations
  - Triangles vs points+triangles
  - Half-edge structure for mesh traversal and editing

- Geometry processing basics
  - Local operations: flip, split, and collapse edges
  - Upsampling by subdivision (Loop, Catmull-Clark)
  - Downsampling by simplification (Quadric error)
  - Regularization by isotropic remeshing

Acknowledgements

- Thanks to Keenan Crane, Ren Ng, Pat Hanrahan, James O'Brien, Steve Marschner for presentation resources